79 May 1983 ($3.00 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems

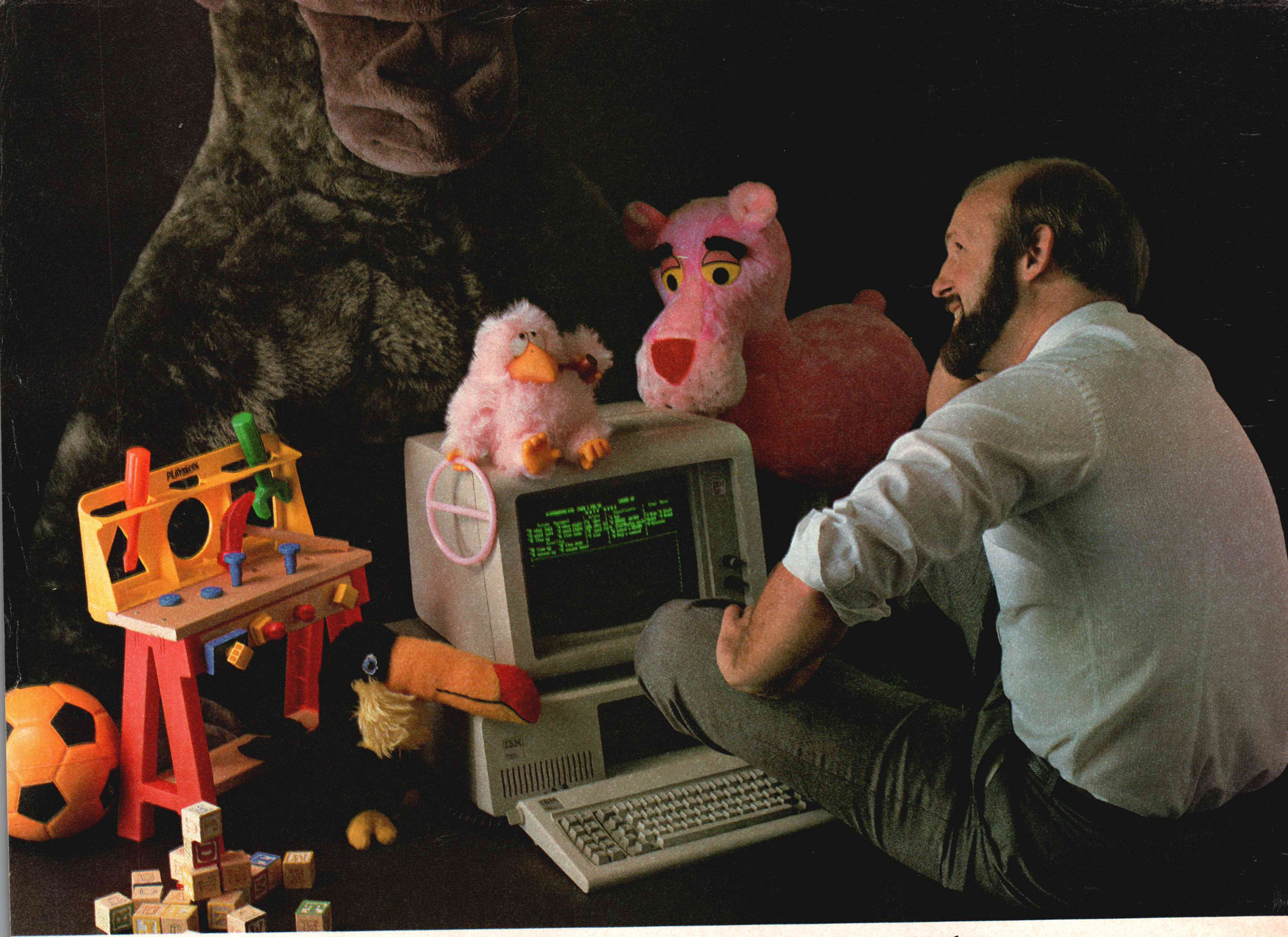
AUGUSTA Part III

A Fast Circle Routine

Shifts and Rotations

SBC, TSX and TXS

Plus: Small-C Screen Editor Enhancements



This Programming professional deserves a lot more from his personal computer.

He's earned it. As a seasoned professional, he's learned to master some of the world's most advanced programming tools. Tools specially designed to meet the everyday demands of programming experts.

But as the owner of a personal computer, he's come to expect less. Less performance. Less sophistication. And less flexibility.

Why should programming a personal computer be any different?

Prior to the announcement of micro/SPF™ development software, experienced programmers felt programming a personal computer was a lot like playing with a toy. You couldn't take it seriously.

But today, there's micro/SPF,™ a solution to elementary program editing tools now offered with most micro-computers.

With micro/SPF™ you get the same procedures and commands experienced programmers are accustomed to using at work. By mimicking features found in standard SPF software, micro/SPF™ provides all the sophisticated utilities programming professionals expect.

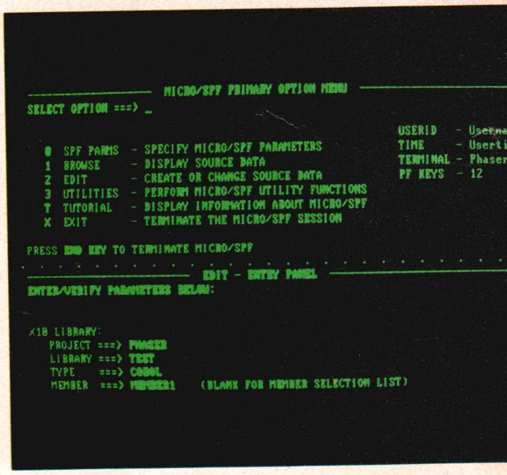
Programming experts can take advantage of skills they've spent years perfecting.

Now, for the first time, mainframe software is available for personal computers. SPF screens are fully reproduced in logical sequence and each screen is formatted identical to those found in the SPF system.

In addition, micro/SPF™ comes equipped with the same primary and line commands, tutorial messages and program editor (with program function keys) experienced programmers are used to.

Programming professionals who've spent years perfecting the art of writing sophisticated code deserve to work with state-of-the-art tools, not toys. Find out how micro/SPF™ can help you do work-compatible programming on your personal computer today!

Circle no. **63** on reader service card.



PHASER

PHASER SYSTEMS, INC 50 WEST BROKAW ROAD
SAN JOSE, CA 95110

Dr. Dobb's Journal

For Users of Small Computer Systems

May 1983 Volume 8, Issue 5

Publisher – Clifford West
Editor – Reynold Wiggins
Managing Editor – Craig LaGrow
Editorial Consultant – Marlin Ouverson
Contributing Editors – Robert Blum, Dave Caulkins, Dave Cortesi, Ray Duncan, Michael Wiesenberg
Marketing Director – Craig Harper
Marketing Manager – Beatrice Blatteis
Advertising Director – Carl Landau
Advertising Sales – Doug Millison
Circulation Manager – Gloria Romanoff
Circulation Assistants – Terri Marty, Linda Marohn
Production Manager – Barbara Ruzgerian
Production Assistants – Jane A. McKean, Alana Hunter
Typesetter – Paula Fairchild
Cover Illustration – Al McCahon

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: DeWitt S. Brown, Burks A. Smith, Robert M. Connors, Robert C. Luckey, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Thomas Davis, Jan Steinman, G. Hunter, Ronald E. Johnson, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S.P. Kennedy, Ed Moran, W.D. Rausch. **Lifetime Subscriber:** Michael S. Zick.

Foreign Distributors **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

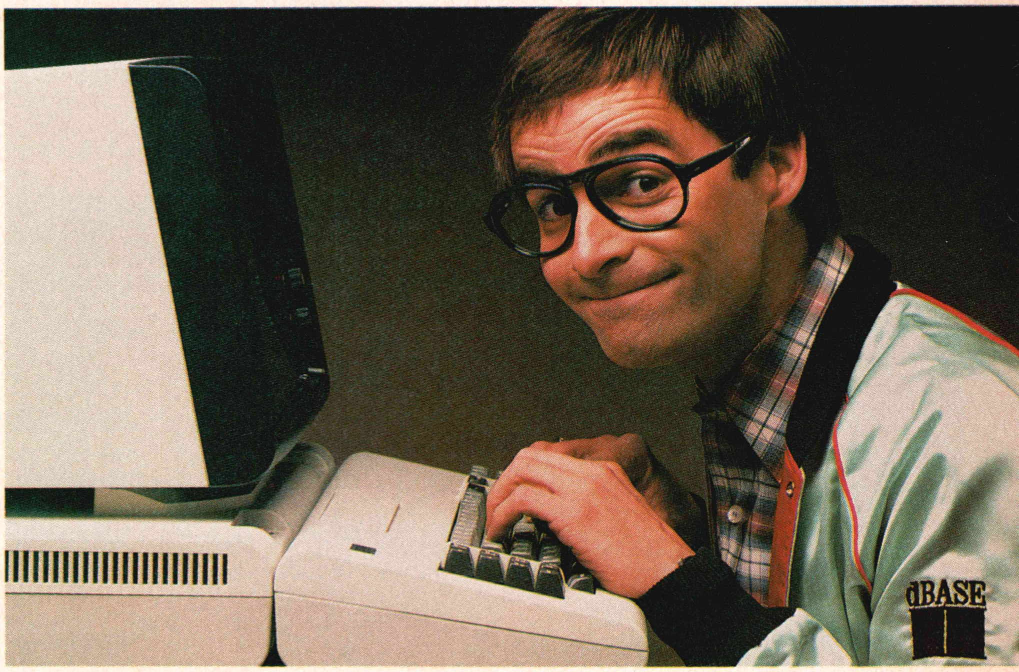
CONTENTS

ARTICLES

DEPARTMENTS

Dr. Dobb's Journal (USPS 307690) is published twelve times per year by People's Computer Company, P.O. Box E, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025 · 415/323-3111

All you dBASE II™ hotshots are about to get what you deserve.



You've written all those slick dBASE II programs.

Business and personal programs. Scientific and educational applications. Packages for just about every conceivable information handling need.

And everybody who sees them loves them because they're so powerful, friendly and easy to use.

But that's just not good enough.

Uh-uh.

Because now you can get the gold and the glory that you really deserve.

Here's how.

We've just released our dBASE II RunTime™ application development module.

And it can turn you into an instant software publisher.

The RunTime module condenses and encodes your source files, protecting your special insights and techniques, so you can sell your code without giving the show away.

RunTime also protects your margins and improves your price position in the marketplace. If your client has dBASE II, all he needs is your encoded application. If not, all you need to install your application is the much less expensive RunTime module.

We'll tell the world.

With your license for the dBASE II RunTime module, we provide labels that identify your program as a dBASE II application, and you get the benefit of all the dBASE II marketing efforts.

We'll also provide additional "how to" information to get you off and running as a software publisher sooner.

And we'll make your products part of our Marketing Referral Service. Besides putting you on our referral hotline, we'll publish your program descriptions and contact information in *dBASE II Applied*, a directory now in computer stores world-wide.

Go for it.

But we can't do any of this until we hear from you.

For details, write RunTime Applications Development, Ashton-Tate, 10150 West Jefferson Boulevard, Culver City, CA 90230.

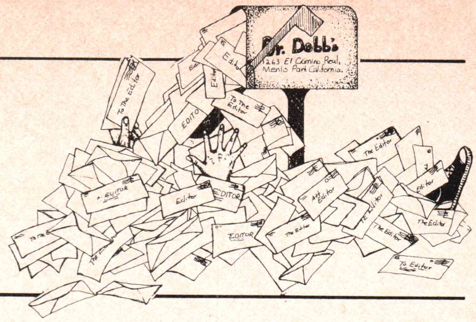
Or better yet, just call (213) 204-5570. And get what you deserve today.

dBASE II™

ASHTON·TATE

Circle **no. 4** on reader service card.

LETTERS



Likes to C It Standard, Too

Dear Dr. Dobb,

I had to jump up and cheer when I read Richard Foulk's letter (Standard Deviation) in *DDJ* No. 77. I am a relative newcomer to C, so when I started seeing deviations from Kernighan & Ritchie, I was dismayed but figured maybe they knew what they were doing. I am glad to see that someone else feels the same way.

I bought Software Toolworks's C/80 which, though not the full implementation, does not deviate from K&R. My hat is off to Walt Bilofsky and his crew. Then I started transferring C User Group software from CP/M to HDSO (Heath) media so I could use it. The hardest part was translating the file-handling functions to the K&R standard to which C/80 adheres. (Are you listening, Leor Zolman? Please bring BDS C back to the K&R fold.)

I always thought that one of the greatest advantages (among several) of C over Pascal was the greater uniformity and portability among versions. Now, that seems to be fading away.

Meanwhile, take a pat on the back for putting out a fine journal, and keep those C features coming.

Sincerely,
Robert L. McClure, Jr.
Route 1, Box 158
Troy, IL 62294

CREF Update

Dear People:

Several people have called with questions about "Cross-Reference Generator in C," in *DDJ* No. 68. The code fragment at the end of line 53 of "cref.c" is actually the end of line 99. The Whitesmith's implementation of alloc() takes two parameters. The first is the number of bytes of space to allocate and the second is put in the first word of the allocated space. All linked lists in the program use the first word in the structure as a link. The whole scheme works very nicely for linked lists implemented this way.

The recursive implementations of lookup() and tree_walk() are elegant, but can use large amounts of stack space. With large files on CP/M-80, this can cause the stack to overwrite the space allocated from the heap. Revised listings are included with this letter (see Figure 1, page 9). Although the revised tree_walk() occupies more space, the capacity of the revised program is larger. Algorithm T of *The Art of Computer Programming*, Volume 1 is the basis of the revised tree_walk(). The stack used in Knuth's version is implemented within the tree being traversed. The left link field is used as a pointer to the node under it on the stack. The tree is destroyed as it is traversed.

One additional trick: when more files are being cross referenced than will fit on a command line, use #include statements on the standard input.

Thank you to those people who called or wrote.

Sincerely,
Jeff Taylor
The Toolsmith
139 G Street, No. 151
Davis, CA 95616

More Benchmarks for the PC

Dear Sir:

We have performed a benchmark test using Fortran and BASIC on several computers. The program was a simulation of the dynamics of a "floating-ring seal" used in high-speed turbomachinery. This is a typical scientific/engineering application, which involves lots of number-crunching. The motivation here was to assess the speed of the IBM PC when used for engineering or scientific applications, and to compare it with other personal and mainframe computers. All tests were performed in single-precision. The results are shown in Figure 2 on page 9 (numbers in parentheses indicate timing with Intel 8087 Math Coprocessor and appropriate libraries).

In a further test, we have run the

EDITORIAL

This issue we welcome two new faces to our ranks. Bob Blum takes over the CP/M Exchange column this month and presents his first installment of a series on CP/M Plus. He will be exploring the features of this new product over the next few months, as well as looking at how to get more from the CP/M 2.2 that you may already have and addressing the usual mailbag items. His knowledge of the ins and outs of CP/M should prove helpful to all of us.

The other new face is Craig LaGrow, our managing editor. Craig is a science writer finishing his master's degree in journalism at Stanford University. With a strong background in publishing and keen interest in computer technology, he will ensure a continued increase in the quality of our publication.

* * *

Many of you have discovered the "quick input" card that appears as an insert with the reader service card. For those of you who have not, let me point it out as an excellent way to drop us a quick note. There is no substitute for a full letter that can be shared, when appropriate, with the readership at large; but this card is a fast, easy way to send suggestions or comments. Either way, we love hearing from you.

* * *

For those who have been wondering, we *are* planning our annual Forth issue. We think you will find it interesting, as always. Further down the road, we would like to publish another telecommunications issue. We have no definite date set, but are looking toward late 1983. Those interested in contributing to an issue on this timely topic should get in touch with us.

On the subject of general contributions, we would like to encourage articles on a wide range of topics. A few that come to mind immediately include the 68xx series of CPUs, 16-bit utilities and applications, C and Unix. If you have an idea or article in an area that might interest our readers, please contact us. We will be glad to discuss it with you.

– Reynold Wiggins,
Editor

LOWEST SOFTWARE PRICES GUARANTEED

PLURIBUS UNUM

We hereby certify that your purchase from Discount Software represents the lowest price sold anywhere. If you find a lower price on what you purchased within 30 days, send the ad and we'll refund the difference.

Discount Price

CP/M

ARTIFICIAL INTELLIGENCE
- Medical (PAS-3) $849
- Dental (PAS-3) $849

ASHTON-TATE
- $4?? dBASE II... call for price
- Financial Planner $595
- Bottom Line Strategist ... $349

ASYST DESIGN/FRONTIER
- Prof Time Accounting $549
- General Subroutine $269
- Application Utilities $439

DIGITAL RESEARCH
- CP/M 2.2
  - Intel MDS $135
  - $149 Northstar
  - $159 TRS-80 Model II (P&T)
  - Micropolis $175
- $98 CBasic-2
- Display Manager $319
- Access Manager $239
- Multiplan $219
- $449 PL/1-80
- BT-80 .... $179
- MAC $85
- RMAC $179
- Sid $65
- $90 Z-Sid
- DeSpool $49
- CB-80 .... $459
- Link-80 .... $90

FOX & GELLER
- Quickscreen $135
- Quickcode $265
- $65 DUtil

MICRO-AP
- S-Basic $269
- Selector IV $295
- Selector V $495

MICRO DATA BASE SYSTEMS
- HDBS $269
- MDBS $1099
- DRS or QRS or RTL $319
- MDBS PKG $1999

MICROPRO
- $289 WordStar
- $199 Mail Merge
- WordStar/Mailmerge $399
- WS/MM/SpellStar $549
- Customization Notes $44
- $199 SpellStar
- DataStar $249
- InfoStar $349
- ReportStar $254
- Wordmaster $119
- Supersort I $199
- Calc Star $129

MICROSOFT
- $229 Basic-80
- $329 Basic Compiler
- $349 Fortran-80
- $549 Cobol-80
- M-Sort $175
- $159 Macro-80
- MuSimp/MuMath $224
- MuLisp-80 .... $174

ORGANIC SOFTWARE
- Textwriter III $111
- Datebook II $269
- Milestone $269

OSBORNE (Mcgraw/Hill)
- G/L, or AR & AP, or PAY ... $59
- All 3 .... $129
- All 3 + CBASIC-2 ...... $199
- Enhanced Osborne $299

PEACHTREE
- G/1,A/R,A/P,PAY,INV(each) $399
- P8 Version .... Add $234
- Peachcalc $249
- Other .... Less 10%

STAR COMPUTER SYSTEMS
- G/L, A/R, A/P, Pay(each) .. $349
- All 4 .... $1129
- Legal or Property Mgt. ... $849

STRUCTURED SYSTEMS
- Business Packages (call)

SORCIM
- $249 SuperCalc
- Act $157

SUPERSOFT
- Ada $270
- Diagnostic II $89
- Disk Doctor $89
- Forth (8080 or z80) $149
- Fortran $319
- Ratfor $79
- C Compiler $225
- Star Edit $189
- Scratch Pad $266
- StatsGraph $174
- Analiza II $45
- Disk Edit $89
- Encode/Decode II $84
- Optimizer $174
- Term II $179
- Utilities I or II $54

SOFTWARE DIMENSIONS/ ACCOUNTING PLUS
- 1 Module $399
- 4 Modules $1499
- All 8 .... $2799

UNICORN
- Mince or Scribble (each). $149
- Both $249
- The Final Word $270

WHITESMITHS
- "C" Compiler $600
- Pascal (incl "C") $850

"PASCAL"
- Pascal/MT+ Pkg $429
- Compiler $315
- SP Prog $175
- Pascal Z $349
- Pascal/UCSD 4.0 ...... $670

DATA BASE
- dBASE II .... Call 4??
- FMS-80 .... $799
- FMS-81 .... $399
- Condor I & III .... Call
- Superfile $159

"WORD PROCESSING"
- Perfect Writer $189
- WordSearch $179
- SpellGuard $199
- Peachtext $289
- Spell Binder $349
- Select $495
- The Word $65
- $145 The Word Plus
- Palantier-1 (WP) $385

"COMMUNICATIONS"
- Ascom $149
- BSTAM or BSTMS $149
- $139 Crosstalk
- $89 Move-it

"OTHER GOODIES"
- Micro Plan $419
- Plan 80 .... $495
- Target PlannerCalc $79
- Target Financial Modeling. $299
- Target Task $299
- Tiny "C" $89
- Tiny "C" Compiler $229
- MicroStat $224
- Vedit $130
- MiniModel $449
- StatPak $449
- Micro B+ $229
- String/80 .... $84
- String/80 (source) $279
- ISIS CP/M Utility $199
- Lynx $199
- Supervyz $95
- ATI Power (tutorial) $75
- Mathe Magic $95
- CIS Cobol $765
- Forms II
- Graph Magic $79
- Basic $249
- Zip MBasic, CBasic $129

APPLE II

ASHTON-TATE
- (See CP/M Ashton-Tate)

BRODERBUND
- G/L (with A/P) $444
- Payroll $355

INFO UNLIMITED
- EasyWriter (Prof) $155
- EasyMailer (Prof) $134
- Datadex $129

MICROSOFT
- Softcard (Z-80 CP/M) $239
- Fortran $179
- Cobol $499
- Tasc $139
- Premium Package $549
- RAM Card $89

MICROPRO
- (See CP/M Micropro)

VISICORP
- Visicalc 3.3 .... $189
- Desktop/Plan II $219
- Visiterm $90
- Visidex $219
- Visitrend/Visiplot $259
- Visifile $219
- Visischedule $259

PEACHTREE
- G/L, A/R, A/P, PAY, (each). $224
- PeachPack P40 .... $395

ACCOUNTING PLUS
- G/L, AR, AP, INV, (each) .. $385

"OTHER GOODIES"
- Super-Text II $127
- Data Factory $269
- Mini Factory $269
- DB Master $184
- Versaform VS1 .... $350

IBM PC, 16 BIT 8," DISPLAYWRITER

"WORD PROCESSING"
- Wordstar $289
- Spellstar $199
- Mailmerge $199
- Easywriter $314
- Easyspeller $159
- Select/Superspell $535
- Write On $116
- Spellguard $189
- Textwriter III $189
- Spellbinder $349
- Final Word $270

"LANGUAGES & UTILITIES"
- Crosstalk $174
- Move-it $129
- BSTAM or BSTMS $149
- Pascal MT+ /86, SPP $679
- CBasic 86 .... $294
- Act 86 .... $157
- Trans 86 .... $115
- XLT 86 .... $135
- MBasic (MSDOS) $329
- MBasic Compiler (MSDOS). $329
  - Both $629
- CBasic Compiler (MSDOS). $495
- Cobol (MSDOS) $649
- Pascal (MSDOS) $429
- Fortran (MSDOS) $429
- "C" (MSDOS) $429
- CP/M 86 .... $239

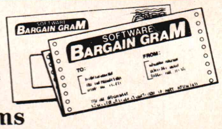
"OTHER GOODIES"
- Lotus 1-2-3 .... $329
- SuperCalc $269
- VisiCalc $219
- Visiplot/trend $259
- Visidex $219
- Easyfiler $359
- Mathemagic $95
- dBase II .... Call 4??
- Condor Q & R, Others .... Call
- Statpak $449
- Optimizer $174
- Desktop Plan $259

FREE WITH PURCHASE:

Complete Software Buyer's Guide ($5.00 value)
Filled with facts and usable advice about scores and scores of software programs from accounting and business systems to word processing and utilities.

Exclusive Service "Hotline"
Our reputation for courteous and knowledgeable service has resulted in calls from people who never purchased our products. Now a separate "hotline" is available to customers only.

Confidential Software BargainGrams



Regular notices of insider's bargains not available to the general public.

DISCOUNT SOFTWARE
Outside Continental U.S.–add $10 plus Air Parcel Post. Add $3.50 postage and handling per each item. California residents add 6½% sales tax. Allow 2 weeks on checks. C.O.D. $3.00 extra. Prices subject to change without notice. All items subject to availability. ®Mfr. trademark. Blue Label $3.00 additional per item. CP/M is a registered trademark of DIGITAL RESEARCH, INC.

ORDER TOLL-FREE VIA VISA OR MASTERCARD:

1 800 421-4003
Calif: 1 800 252-4092
6520 Selma Avenue, Los Angeles, CA 90028

DD 583

DISCOUNT SOFTWARE

Figure 1.

```
/* lookup - find pointer to node for "name" */
struct node **lookup(root, name)
  register struct node **root;
  register char *name;
  {
  register int c;

  while(*root && (c = lexcmp(name, (*root)->name, 0)))
    root = c < 0 ? &(*root)->left : &(*root)->right;
  return root;
  }

/* tree_walk - walk a binary tree, doing ftn to each node */
tree_walk(root, ftn)
  register struct node *root;
  int (*ftn)();
  {
  register struct node *stack, *tmp;

  stack = NULL; /* stack initially empty */
  for ( ; ; )
    if (root) {
      tmp = root;        /* move to the left */
      root = root->left;
      tmp->left = stack;         /* stack node */
      stack = tmp;
      }
    else if (stack) {
      root = stack;      /* pop stack */
      stack = stack->left;
      (*ftn)(root);
      root = root->right;
      }
    else
      break;
  }
```

NBS Methane properties package on the IBM-PC under Fortran 3.03. Execution time was 16.5 seconds without the 8087, and 2.5 seconds with the 8087. A time of 90 seconds was obtained for this program on the Xerox 820.

Two results deserve special comment. First, both the original IBM release of Microsoft Fortran and the new 3.03 release (presently under Beta Test) are laughably slow. Note that even the 3.03 version is practically as slow as the 8-bit Fortran of the Xerox 820! This result is presumably due to Microsoft's unwise decision to write their Fortran in Pascal – thus burdening it with all of Pascal's inefficiencies. Even Microsoft's own BASIC compiler is faster! These Fortrans are therefore nearly useless as serious tools for engineering work, and we can only hope that some enterprising firm will fill the need for a fast Fortran for the IBM PC.

Next, the BASIC compiler with the 8087 does not achieve anything like the speed improvement that Fortran does. Whether this is due to inefficiencies in the MicroWare BASIC87 library or to problems with the BASIC compiler itself is an open question.

Sincerely,

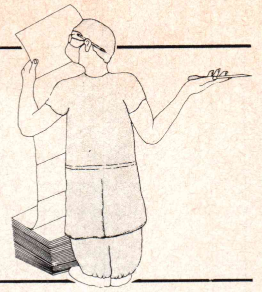
Jim Glass
Chris Landis
4747 Orion Avenue, Apt. C
Sherman Oaks, CA 91403

DDJ

Figure 2.

Machine	Language	Approx. Execution Time Seconds	
CDC-176	FORTRAN	0.8	
IBM-3033	FORTRAN	1.63	
UNIVAC 1100/83	ASCII FORTRAN	12	
TEKTRONIX 4081	FORTRAN	196	
PDP-11/34	FORTRAN	94	
IBM-PC	BASIC Compiler	866	(773)*
Xerox-820	FORTRAN	3386	
IBM-PC	IBM FORTRAN (original)	4440	(1711)*
IBM-PC	FORTRAN 3.03 (new Microsoft release)	3284	(585)**
IBM-PC	BASIC Interpreter	11935	
Apple	BASIC Compiler	5153	
Apple	BASIC Interpreter	16937	

* Using MicroWare 8087 Library
** Using Microsoft 8087 Library



by D.E. Cortesi, Resident Intern

Disk Problems, Continued

Back in February we presented Loren Amelang's problem of backup diskettes that would often be unreadable after a few months of storage. Randolph Fritz, of Mahwah, New Jersey, writes with a number of suggestions. "Problems like Amelang's," he says, "are often caused by mechanical disk problems. Loren's drives might just have dirty heads. If so, the normal decay of disk magnetization, most of which takes place within a few days of writing, could make reading erratic.

"Then, if there are two drives, they may be subtly incompatible. This is especially likely if one is used as a system disk and the other to write backup copies. Finally, the disk drive's head positioner may be worn. In this case, track positioning becomes erratic and dependent on the order in which tracks are read and the rate at which tracks are read."

The net of Fritz's suggestions is that Amelang's disk drives should be taken to a competent repair shop for cleaning, inspection, and alignment. He gives some suggestions on how to do it yourself, but ends by saying, "Disk drive maintenance is a complex subject. Perhaps one of your readers would suggest a good book on the matter?" Indeed. Is there a book, or a manufacturer's manual, that has a good description of how to do routine maintenance on disk drives? Or is there a drive mechanic in the house who'd like to do an article on the subject?

Disk Format Survey

Aubrey Hutchison wants to know if anyone has ever gathered and published a listing of the disk formats used by different machines. We don't know of any; does a reader? If not, it would be neat to publish a really comprehensive list, especially of five-inch formats, in *DDJ*. Why not send us a description of the disk format(s) supported by your system, something like this.

tracks	: 0..39
sectors/track	: 1..10
bytes/sector	: 512
skew sequence	: 1,4,7,10,3,6,9,2,5,8
DPB.OFF	: 2 (reserved tracks)
DBP.DRM	: 3F (64 directory entries)
DPB.DSM	: C5 (198 data blocks)
blocksize	: 1024

The last four items apply only to CP/M, of course; they are the numbers from the Disk Parameter Block which determine how the BDOS allocates space on the drive. If you don't use CP/M, say what the OS is, and include whatever software values are important or would vary from disk to disk.

Hang 'em High

Another continuing problem is how Ernest Knipp's Z80 could get hung up so high that he has to power the system down and up to restart it. Don Taylor recalls a similar problem when he built "a homebrew disk interface with a DMA controller. Every time I tried to do a transfer, the machine would lock up and reset would not help. It turned out that the BUSACK signal was not getting to the controller, but the CPU had shut down waiting for the controller to release BUSRQ. BUSRQ, it seems, has higher priority than a reset signal."

Alan Behler of Newport, North Carolina, has another suggestion. He passes on a tip from an early edition of *Computer Notes*, a newsletter published by MITS for the Altair: "Hold the RUN/STOP switch on the front panel to STOP, and at the same time press RESET." Behler says that the same tip applies to the IMSAI 8800. The problem seems to be a conflict between the CPU and the front panel hardware.

Controlling MBASIC

A while ago, we passed on Aubrey Hutchison's tip that control-A could be used to make MBASIC duplicate a line of a program. William McWorter explored it and found that "it appears that you can edit whatever happens to be in the input buffer at the time control-A is entered. Thus you can re-execute an immediate line by typing control-A followed by E."

Harvey Hahn knew more about control-A. He writes, "I have been using the control-A technique for a number of years with my Altair 8800. It was explained in a full-page article in *Computer Notes*, Volume 3, Number 7 (Jan/Feb 1978), page 18." Hahn sent along the lead paragraph of that article:

"CTRL-A, a feature of MITS® BASIC, has become a powerful programming aid, due to an undocumented feature discovered by Donald Fitchhorn of MITS. If CTRL-A is typed immediately after EDITing a program line, the edited line is returned as a command to be edited. Thus CTRL-A can be used to shuffle program lines, break apart multiple statement lines, and isolate program errors."

Undocumented in 1978, and undocumented still. Oh, well.

The Buffered Keyboard

Our shiny new BIOS is now working. Absolutely its best feature is an interrupt-driven, buffered keyboard. When a key is pressed, an interrupt occurs, and the input byte is saved in a ring buffer. This cures a problem that has plagued several of our favorite applications.

Our H19, like many terminals, generates a two-byte sequence when one of its function or cursor-move keys is hit. The first byte is ESC; the second is a printable character. An application like SuperCalc can be customized to handle these sequences, and as long as keys are hit sloooowly, everything works fine. But if a special key is hit twice, rapidly, the program may not have finished its response to the first key when the second key's bytes start to arrive. The ESC is replaced in the I/O port's buffer by the second character, and is lost. When the program finally gets around to polling the console, it finds only the printable character. Among our programs, SuperCalc was the worst afflicted, because it updates several places on the screen whenever a cursor-move key is hit.

The interrupt-driven BIOS cures the problem for all programs, because it takes control within microseconds of a byte's arrival and puts it away for later use. Now we can run off a string of cursor moves with the repeat key, and sit back to watch SuperCalc huffing and puffing to catch up. But it wasn't easy to get this feature. We had to make several hardware fixes.

CCS vs. CompuPro

The first problem was that our CCS 2810 Z80 board would not work with the interrupt controller on our CompuPro System Support board. The CompuPro board monitors the vectored interrupt lines of the S-100 bus and generates a single INT signal when one is pulled. The CCS board would recognize INT and go into an interrupt acknowledgement cycle, but what it fetched from the data lines was always an RST 7, not the three-byte call instruction that the CompuPro was supposed to generate. Reset 7 is FFh, and it really indicated that there was no data at all on the bus at the time the CPU was looking for it.

Let's skip lightly over several hours of fairly intense work and move to the

conclusion. The CCS 2810, or at least our elderly example of it, was not kosher by IEEE 696 standards. It was generating sINTA, the interrupt acknowledgement, much too late in the interrupt cycle for the CompuPro board to respond (two clocks later than pSYNC, in fact). Oddly enough, it *was* generating a perfectly good INTA status signal; however, this signal is gated onto the data bus along with the other simulated 8080 status bits, as a compatibility feature. The CompuPro, as you might expect, wouldn't have anything to do with such non-IEEE frippery. It expects sINTA to become valid, along with the other S-100 status bits, when pSYNC comes up. We were able to get the good INTA signal onto the sINTA bus line by cutting one trace and soldering in one jumper on the CCS board.

(That sounds so cool. Just cut a trace, solder a jumper, pooh, no problem. Listen, no diamond cutter ever braced as carefully as we did before cutting that trace. And the sweat was rolling in streams when we powered up afterward. Would our one and only CPU board come up, or would some chip quietly go phhht and leave us without a computer? And it did *not* come up! *PANIC!* Then, looking down from the ceiling, we noticed that we had forgotten to connect the ribbon cable that runs from the CPU board to the CRT. With that hooked up, it ran. Phew.)

CompuPro vs. Itself

We were using one port of our old CompuPro Interfacer II board as a source of test interrupts. A Diablo printer with a keyboard was hooked to it, so we could produce a byte and an interrupt, while a test program reported the results on the CRT. The Interfacer II generated interrupts just as expected, provided we enabled only received-data interrupts. If we enabled transmitted-data interrupts, the test program would report a continuous stream of them, indicating that the Interfacer II was holding its vectored interrupt line in a low state all the time.

This should not have been the case. According to the schematic, the Interfacer II should reset its interrupt signal when it sees sINTA. We pulled the board to see if there was a chip we could change easily. Then the problem became clear: the board didn't match the schematic that came with it. Where there should have been a chip U13, there was only blank green board; the inverted Transmit Buffer Empty signal from the UART went straight to the interrupt line and there was no reset circuitry at all! We never did find out whether we got an early board with a late schematic, or vice versa. The Interfacer II isn't in production anymore, and thank goodness we didn't need its interrupts.

CCS vs. The World

The interrupt we really wanted was one from the UART on the 2810 CPU board, to which the CRT is attached. Unfortunately, CCS designed their board assuming that only polled I/O would be used. They used the nice 8250 UART (the IBM PC uses it, too), which has an elaborate structure of prioritized interrupts built into it. However, CCS left the 8250's interrupt pin unconnected. We'd inquired about this when first looking into interrupts a year ago, and were then given instructions on how to access the 8250's interrupt ability. Here, for the information of those who have a 2810 CPU board, is how to do it.

The 8250 generates an interrupt signal on its pin 30. This is a positive-going, CMOS signal. It has to be buffered and inverted before it can be applied to the S-100 bus. Near it on the board is U7, which contains an unused XOR gate (make sure it really is unused); this can be used as an inverter. We were able to connect U5 (the 8250) pin 30 to U7 pin 12, and U7 pin 13 to U7 pin 14 (tying it to +5 volts). Then we ran a rather lengthy jumper from U7 pin 11 up and over the top edge of the board, and all the way down to the solder pad for S-100 line 4, VIO*.

It worked perfectly the first time. During a cold start, our BIOS has to prime

the CompuPro interrupt controller (it takes hours of study to figure out which of the 8259's many options to program) and also load the 8250's interrupt control register to allow received-data interrupts. Information on 8250 interrupts has been omitted from the CCS manual; you have to have the 8250 data sheet (or use the Technical Reference manual for the PC, which we had). We also allow the 8250 to interrupt when the Break key is hit. Our BIOS takes this as a signal to discard any buffered input, but it could be used as a hyper-escape into a monitor of some kind.

Software vs. Common Sense

Then we worked out the software problems of using a buffered keyboard. The first was the CCS disk formatter, whose second instruction was a DI. That made it hard to respond to its prompts for keyboard input. But a stickier problem arose.

If the keyboard is buffered, when is a byte of input "ready"? The BIOS has to answer this question in its routine for Console Input Status. The simple answer is for the BIOS to report "ready" whenever the keyboard buffer is not empty. That's the answer we were forced back to after trying more clever schemes.

If the simple answer is impemented, then you can't use the keyboard buffer to type ahead of the system. It would be nice if you could type one PIP transfer, say, while PIP was working on the previous one. But if that PIP transfer is to a character device like LST:, PIP checks the console input status after every line of output and aborts if a key has been hit. So you can't type ahead of a PIP to LST: because PIP will quit.

Things get worse. The BDOS polls the console status routine every time it sends a byte of output to the console. If an input byte is ready, the BDOS reads it and tests to see if it is a control-S. If it is not, the BDOS tucks it away in a one-byte buffer. If you try to type ahead of *any* command that does console output, you will lose one byte of typed-ahead material for every byte of output the program displays. For example, you might start PIP on a disk transfer:

```
A>pip b:=a:somethin.dat
```

and while it is working, you type "stat b:*.*" as your next command. But when PIP ends, the two-byte prompt "A>" is typed. Before writing the "A" the BDOS polls the console and reads your typed-ahead "s." Before writing the ">" the BDOS reads your typed-ahead "t" and stores it over the "s." What appears on the console is

```
A>tat b:*.*
```

Our first version of buffered input used a smarter solution. The BIOS reported that console input was "ready" only when there was data in the buffer *and* the first byte of data was a control character or DEL. That worked, in that it kept the BDOS from swallowing typed-ahead bytes. And it allowed us to use control-S to hold output, control-Q to release it, and DEL to cancel a submit file. It would have been the perfect solution if it weren't for the foolishness of certain programmers.

In two major packages that we know of, the code for console input looks like this:

```
repeat
    Test console status
until console is ready;
read console input
```

We cannot comprehend why anyone would do this. In a polled system, it is needlessly complicated, but it works. With a buffered keyboard it is a disaster. The program won't read until the console is ready, but the console shouldn't appear ready until a control character is at the head of the buffer. The result is that when one of these programs starts up, the system dies. It took a little time to figure out what was going on, the first time. We patched that program. Then the second one turned up, and we gave up and went back to the simple answer for console input status.

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 227**

AUGUSTA, Part III

Recursive Descent Compilers

Compilers are computer programs that perform language translation. Typical compilers translate a high-level lanugage, such as BASIC or Pascal into the machine language of the computer that the program will run on (see Figure 1). Part III of this four-part series describes how one type of compiler works.

Compilers as Language Translators

The compiler doesn't have to translate directly into machine code. Some compilers output assembly language that must be assembled later. Others output an intermediate language that, in turn, is compiled to machine code or interpreted at run-time. There are even compilers that translate from one high-level language into another. For example, the RATFOR language is compiled into Fortran.

Augusta programs are compiled into an intermediate language called "pseudocode" ("p-code" for short). In effect, the p-codes are the machine language of a hypothetical or pseudo-machine. The compiler uses a technique known as "recursive descent parsing" to perform the translation. Compilers, and in particular recursive-descent compilers, are the subject of this article. Included in this part is the first half of the Augusta compiler source listing (see page 20). Part IV contains the rest of the compiler listing and a detailed description of the compiler's operation.

The Compilation Process

Compilation is divided into three steps: (1) lexical analysis, (2) syntactic analysis, and (3) semantic analysis and code generation. Some compilers make several passes through the source program, perhaps performing one of the above steps on each pass. Augusta is a one-pass compiler, so all three steps occur simultaneously.

Lexical analysis carves the source program into logical entities called "tokens." To the computer, a source file is just a long sequence of characters. The lexical analyzer breaks the characters into recognizable groups such as "IF", "THEN", "+", "(", "ALPHA", and so on.

Syntactic analysis or "parsing" ensures that the tokens returned by the lexical analyzer appear in the correct order. The syntax diagrams presented in Part I of this series describes the order of tokens for all valid Augusta programs.

When the syntax analyzer recognizes a valid sequence of tokens, the compiler outputs p-codes. During compilation, these three steps are intermingled. When the syntax analyzer needs a new token, it calls the lexical analyzer. When enough tokens have been read so that the parser can recognize a statement or phrase of the language, then the code is generated.

Lexical Analysis

The lexical analyzer reads the source program character by character. Once it recognizes a valid sequence of characters, it returns the entire group as a token.

The classical approach to designing a lexical analyzer is to draw a *state diagram*. The state diagram is really just a special kind of flowchart where circles replace the usual rectangular boxes and diamonds. The diagram describes what the lexical analyzer should do for each character that it sees. Figure 2 (page 14) shows a diagram to separate digits from letters in a stream of characters like "ABC5347 PWX943ZW1F0". That input would produce the following sequence of tokens: "ABC", "5347", "PWX", "943", "ZW", "1", "F", and "0".

Each circle inside the diagram represents a *state*. The diagram begins with state 1. On seeing the letter "A", the lexical analyzer reads a new character from

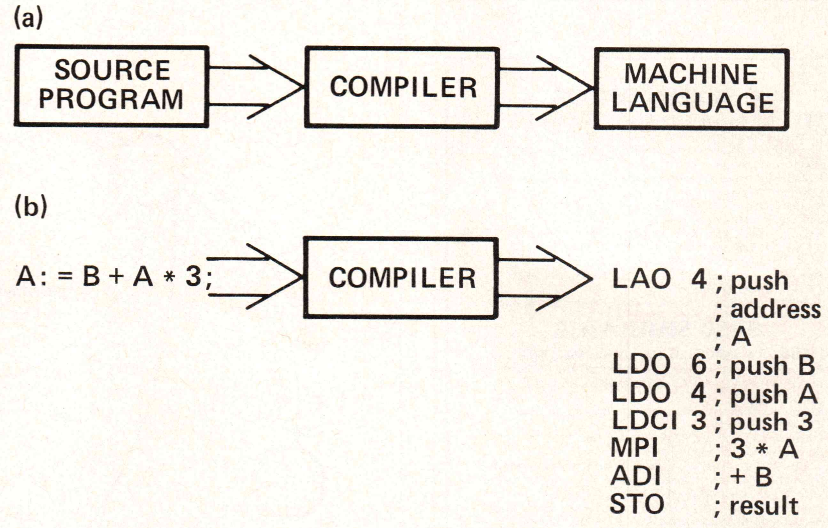


Figure 1.
(a) A compiler translates a high-level language into the machine language of the computer that the program will run on. (b) The translation of a simple Augusta statement into p-codes.

by Edward Mitchell

Edward Mitchell, Box 390145, Mountain View, CA 94039.

the input and jumps to state 3. The asterisk ("*") indicates that a character is read *before* entering the new state.

The analyzer remains in state 3 as long as it continues to see an alphabetic character. Upon reading "5" from the input, the analyzer jumps to state 4. At this point, it has identified the first complete token "ABC". Returning to state 1, the analyzer is still looking at a "5" so it jumps to state 2 and processes the digit characters. Listing 1 (page 17) shows how the state diagram can be converted to a BASIC program.

A compiler, such as Augusta, must recognize a large number of tokens – many more than just letters and numbers – for example, all of the arithmetic operators, "+", "-", "*", and "/", plus all of the punctuation symbols like "(", ";", ")", and ":=". Some symbols are harder than others to recognize. For example, "/" by itself is the division symbol. But if it is immediately followed by "=", as in "/=", it becomes the "not equal" relational operator. Similarly, some tokens can be quite lengthy. Take the character string, for example. It begins with a double quote (") followed by a large number of characters that are terminated by a trailing quote.

Just as flowcharts are unnecessary for coding many programs, state diagrams are not really needed for many lexical analyzers. Instead, a practiced programmer can directly code an appropriate analyzer fairly quickly. To wit, Augusta's lexical analyzer was written in a single evening and appears in lines 1280 through 1775 of Listing 2 (page 20).

The lexical analysis stage of the compiler will often do additional work to aid the rest of the compilation process. Numbers are converted from their character representation (e.g., "197") into an internal numeric form (e.g., TN=197, where TN is a variable). All identifiers (any token beginning with an alphabetic character) are looked up in the keyword table. If the identifier is a keyword like "AND" "PROCEDURE", or "ELSE", then the lexical analyzer marks the token as a keyword. Unquoted lower case letters are converted to upper case.

In most compilers, the lexical analyzer is a subroutine. At each call, it returns the next token from the program source.

Syntax Analysis

The syntax analyzer or "parser" checks that the tokens appear in the proper order. The "proper order" is determined by the language's syntax which is the set of rules that defines the language. The syntax diagrams in Part I (January 1983) of this series show what valid Augusta statements look like.

How does syntax analysis work? There are actually several techniques for parsing a computer program. Augusta uses a method called recursive-descent parsing, which is a variation of "top-down" parsing. The names "recursive descent" and "top down" come from how the parser builds a *parse tree*. (You don't need to understand the theory in great detail. However, it's convenient for following the basic operation of the parser. I'll keep this theoretical section to a minimum.)

A tree is a special type of data structure that is normally depicted upside down as in Figure 3 (page 15). This implies that the "root" is at the top of the tree and the "branches" are at the bottom. Top-down parsing is so named because it produces a tree-like structure from the top and going downwards as it parses a statement. Compare the syntax diagrams in Figure 4 (page 16) to the parse tree in Figure 3(b) (page 15). The parse tree represents the path that the compiler follows through the syntax diagrams as it parses an expression.

To see how syntax analysis works, we'll construct a parser for the simplified expressions described by the syntax diagrams in Figure 4 (page 16). These diagrams recognize the algebraic hierarchy of operations: "*" and "/" come before "+" and "–".

The syntax diagrams are recursive because one of the definitions is defined in terms of itself. EXPR is defined using SE; SE is defined using FACTOR; and in the case of a parenthesized expression, FACTOR is defined using EXPR.

The diagrams can be translated into a set of procedure calls. A simple implementation of the parser written in Augusta is shown in Listing 3 (page 27). Procedure GETTOKEN (not shown in Listing 3) reads a single token of input and places it in the global variable TOKEN. TOKEN is a character containing "C" if a constant was read, "I" if an identifier was read, or

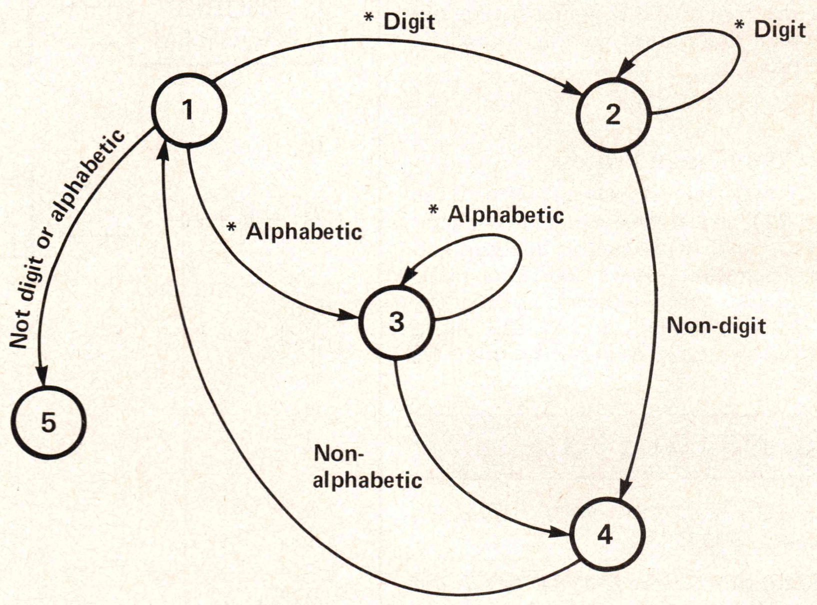


Figure 2.
A simple state diagram to separate letters from digits. See text for details.

one of the punctuation symbols "(", ")", "*", "/", "+", "-".

To parse the expression, 3*(5+6), GETTOKEN reads the "3". EXPR calls SE to evaluate a "simple expression." SE, in turn, calls FACTOR. Seeing a constant "C", FACTOR reads the next token and returns to SE. Since the token "*" is not one of "+" or "-", control returns to EXPR. But "*" is recognized by EXPR, so EXPR reads the next token, a left parenthesis "(", and calls SE again. SE calls FACTOR where the parenthesis is handled. FACTOR scans past the "(" by calling GETTOKEN and calls EXPR to evaluate the subexpression. After returning from EXPR, FACTOR sees the right parenthesis ")" and returns to SE, which returns to EXPR.

Semantic Analysis

Statements that are syntactically valid may still not make sense. Variable identifiers such as A and B can be multiplied together if and only if A and B are numeric variables. If A and B are strings, then the statement is meaningless. For example,

(1) A := "String 1";
(2) B := "String 2";
(3) C := A * B;

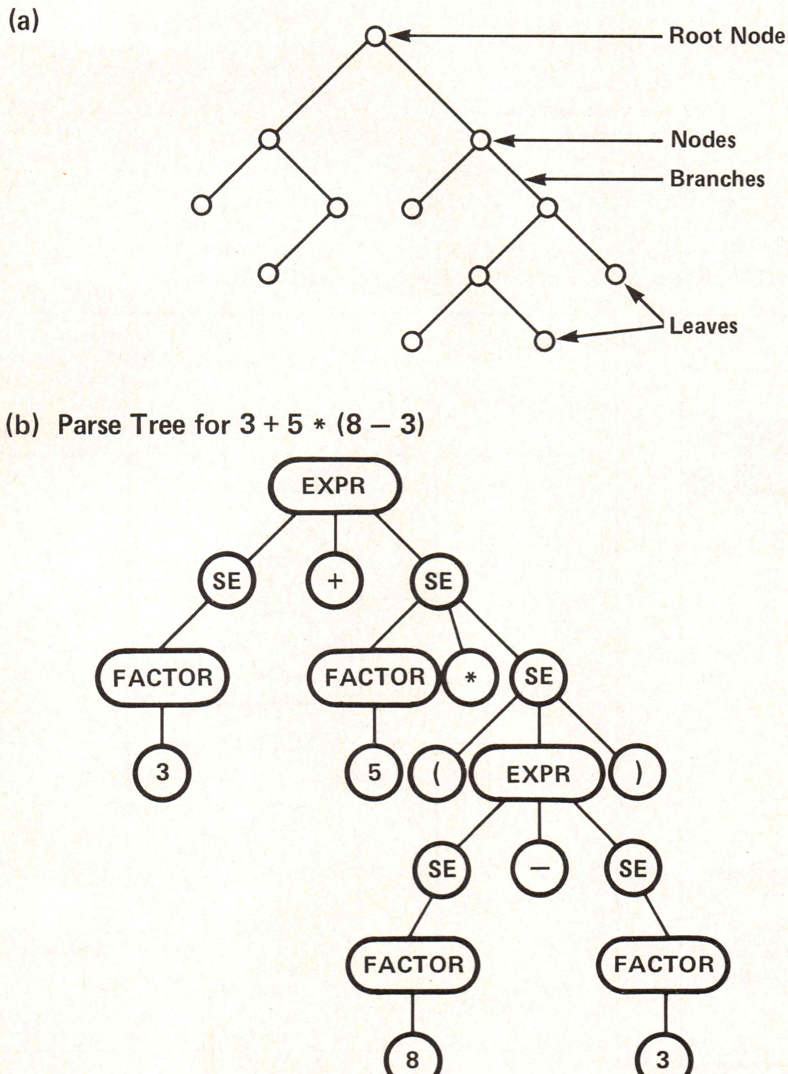


Figure 3.
(a) shows a data structure called a *tree*. Because the tree is depicted upside down, the root is on the top. (b) shows a *parse tree* produced by parsing a simple expression according to the syntax diagrams in Figure 4. Parse trees are the basis of recursive-descent parsing, the technique used by the Augusta compiler.

Statement 3 is syntactically correct but semantically wrong.

Semantic analysis attempts to infer meaning from the statement; e.g., C is to be assigned the quantity of A multiplied by B. Once the compiler understands the statement, it can generate the corresponding p-code translation.

Syntax analysis only cares about finding the correct tokens, such as a constant, an identifier, a keyword, or so on. Semantic analysis, on the other hand, examines the value of the tokens. For example, the identifier "ALPHA" must be looked up in a symbol table to see if it is defined. If it is defined, then the compiler must determine that it can be used correctly in the current context.

Code Generation

Augusta generates code as it does the parsing. An example, using the simple expression parser, shows how this is done.

Let

```
X := 5 * 4;
```

be a simple assignment statement. The compiler, upon seeing an identifier followed by ":=" begins parsing the assignment statement. The symbol "X" is looked up in the symbol table, which tells the compiler that X is an integer variable stored at global offset 10. Augusta generates

```
LDA 10 ; Load address of variable X
```

The compiler next scans across the expression. But note that because Augusta generates code for a stack machine, both operands (the "5" and the "4") must be pushed on the stack before performing the multiplication. This means that the following sequence of p-codes will be created:

```
LDCI 5   ; Load constant 5 on to the
         ; stack
LDCI 4   ; Load constant 4 on to the
         ; stack
MPI      ; Pop top 2 words, multiply
         ; together, and push result
```

Finally, the top of stack value is stored at X, by emitting the instruction

```
STO      ; Store top of stack value in-
         ; to the address contained in
         ; the next word on the stack
```

Listing 4 (page 28) shows the simple expression parser modified to generate p-codes.

Two new procedures, EMITBYTE and EMITWORD, have been created. The parsing routines call EMITBYTE and EMITWORD when they need to output p-codes or p-code parameters. For example, the LDCI opcode is output with the following calls:

```
EMITBYTE (1)  ; – LDCI p-code
              ;   value
EMITWORD (5)  ; – constant 5 fol-
              ;   lows LDCI
```

Code Generated for FOR Loops

The previous section gave an overview of the code generation scheme. We will now look at some specific statements and see what the corresponding p-code translation looks like.

The FOR loop has the form

```
FOR <Identifier> IN
   <Starting Value> . .
   <Ending Value>
LOOP
   <Sequence of Statements>
END LOOP;
```

For example,

```
FOR I IN 1 . . 10
LOOP
   PUTINT(I);
   NEWLINE;
END LOOP;
```

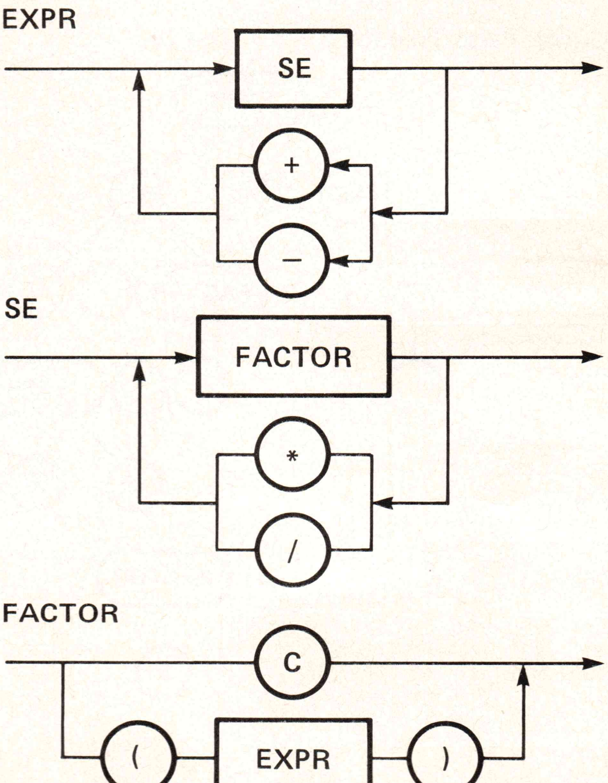



Figure 4.
Syntax diagrams for a simple expression parser.

prints the integers 1 throught 10.

For the loop above, the compiler must emit code to set I:=1 at the beginning of the loop to check to determine if I is greater than 10 and to increment I each time the loop is executed. Listing 5 (page 29) shows sample code for the loop shown above.

When the FJP instruction at (1) in Listing 5 is emitted, the compiler does not yet know where the program should jump to. Until the code for the enclosed sequence of statements is generated, the compiler doesn't know how many bytes the FJP instruction should jump past. Instead, it must remember where the FJP instruction is located, generate the remaining code, and then go back and fix the FJP instruction so that it will jump to the correct location. This technique is called "back patching" and is used extensively when compiling long sequences of IF-THEN-ELSEIF and CASE statements.

The WHILE Statement

The WHILE statement is translated into a conditional test, followed by the enclosed statements and ending with a jump back to the conditional test. Listing 6(b) (pages 28 and 29) shows the code produced for WHILE statement shown in 6(a) (page 28).

IF-THEN Statements

Listing 7(a) (page 29) shows sample code for a simple IF-THEN-ELSE statement. Back patching is done for all the forward jump instructions, which includes both the FJP and UJP p-codes in Listing 7(b) (page 29).

"Short circuit" boolean expressions, whether used in assignment statement or as part of an IF-THEN test, use the FJP instruction to jump over portions of the expression code. Listing 8(b) (page 30) shows the code generated for the statements in 8 (a) (page 30).

The CASE Statement

The CASE statement is unusual in that it uses the XJP (indirect jump) opcode to branch to the appropriate section of code. XJP is a seven-byte opcode, having the format

XJP w1, w2, w3

where w1, w2, and w3 are 16-bit words. XJP uses the value on the top of the stack as an index into a jump table that appears after the code for each of the cases. w1 and w2 are the minimum and the maximum case selections, respectively. For example, in a CASE statement like

```
CASE I IS
   WHEN 7 => ...
   WHEN 11 => ...
   WHEN 19 => ...
   WHEN 23 => ...
END CASE;
```

the minimum selector, w1, is 7, and the maximum selector, w2, is 23. Word w3 is the offset to the indirect jump table. Listing 9 (page 30) shows the basic format for the case statement.

The jump table has the following format,

```
→ UJP          instruction
  address of OTHERS condition
  address of case w1
  address of case w1+1
  address of case w1+2
        .
        .
        .
  address of case w2-1
  address of case w2
```

For the CASE statement shown previously, with selectors 7, 11, 19, and 23, an entry is made in the jump table corresponding to each selector. But the jump table has sixteen entries (maximum selector minus minimum selector or 23 - 7) and so only four entries are accounted for. All other, unused selectors are filled with the address of the UJP instruction at the beginning of the table. Then, when a case value for I is not one of 7, 11, 19, or 23, the program does an automatic jump to the OTHERS clause. The first entry in the table is the address of the code to handle the OTHERS condition. In the event that there is no OTHERS clause, this entry simply jumps to the first instruction after the jump table.

In the July Issue

Part IV contains the final half of the Augusta compiler source listing. It concludes with some personal comments on the subjects of Ada, the rationale for creating Augusta, the DoD, and Microsoft BASIC. DDJ

(Listing One below)

(Listings 2 through 9 on pages 20 through 30)

Augusta Part III

Listing One

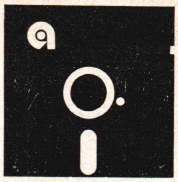
```
10 'SIMPLE LEXICAL ANALYZER
20 '
30 DIGITS$="0123456789"
40 ALPHA$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
50 INPUT "ENTER STRING:   ",S$
60 P=1              ' P POINTS TO NEXT CHAR TO READ
70 GOSUB 800        ' READ CHAR INTO CH$
100 'STATE 1 - DECIDE IF ALPHABETIC OR DIGIT
110 IF INSTR(DIGITS$,CH$) THEN GOSUB 800: GOTO 200
120 IF INSTR(ALPHA$,CH$) THEN GOSUB 800: GOTO 300
130 GOTO 500
200 'STATE 2 - READ IN DIGITS
210 IF INSTR(DIGITS$,CH$) THEN GOSUB 800: GOTO 200
220 GOTO 400
300 'STATE 3 - READ IN ALPHABETIC CHARS
310 IF INSTR(ALPHA$,CH$) THEN GOSUB 800: GOTO 300
320 GOTO 400
400 'STATE 4 - OUTPUT THE EXTRACTED TOKEN
410 PRINT TOKEN$
420 TOKEN$=""
430 GOTO 100
500 'STATE 5 - INVALID CHARACTER STOPS
510 STOP
800 'READ CHARACTER OF INPUT
810 IF P>LEN(S$) THEN CH$=".": RETURN
820 TOKEN$=TOKEN$+CH$     ' ACCUMULATE A TOKEN
830 CH$=MID$(S$,P,1)      ' GET NEXT CHARACTER
840 P=P+1
850 RETURN
```

End Listing One

(Listing Two begins on page 20)

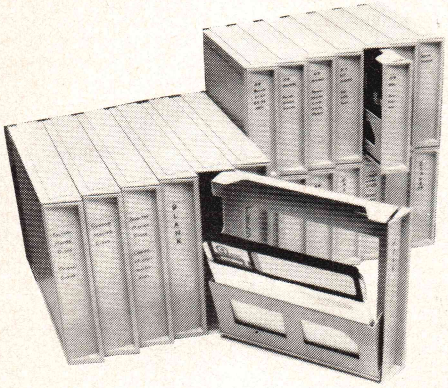
DISKS

Lifetime certification. Hard holes for reliability. Sold in boxes of 10.



	Single Side Single Den.	Single Side Double Den.	Double Side Double Den.
5¼″ *	— —	$26.95	$31.95
8″	$29.95	34.95	42.95

*Specify soft sector or 10 or 16 hard sectors.



DISK BANK

Versatile modular disk filing system stacks vertically or horizontally. Each case holds 10 disks.
5¼″ case **$6.95**
8″ case **$7.95**

VISA and MASTERCARD accepted. California residents add 6% sales tax. Telephone orders (213) 661-2031. Please add $2 per item shipping.

CDE CALIFORNIA DIGITAL ENGINEERING
P.O. BOX 526 ★ HOLLYWOOD, CA 90028

Circle **no. 10** on reader service card.

LEO ELECTRONICS, INC.
8921 S. Sepulveda # 208
Los Angeles, CA. 90045
(213) 641-3101 (800) 421-2418
TLX: 664-688 Interline LSA

LOOK!!

LOW PRICES

RAMS

4116 (200ns)	1.10
4116 (150ns)	1.25
4164 (200ns)	4.65
4164 (150ns)	4.85
2114 (200ns)	1.50
6116P-3	4.25

MICROPROCESSORS

Z-80A	4.50
8080A	2.75
8085A	6.00

EPROMS

2708	2.50
2716	3.20
2716-1	5.00
TMS 2716	4.75
2732	3.95
2532	4.50
2764	8.00

REGULATORS

ALL .75
TO—220

TERMS: Check, Visa, Mastercard. Call for C.O.D. U.S. Funds only. California residents add 6½% sales tax.
SHIPPING: Add $2.00 for Ground and $5.00 for Air.
ALL MAJOR MANUFACTURERS
ALL PARTS 100% GUARANTEED

Circle **no. 48** on reader service card.

NEW

CodeSmith™-86

The debugger with the most bite for your IBM PC!
With unique multi-window/multi-level split screen features:

- Full line-edit keyboard utilization—insert, delete, next-word, previous-word, etc.
- Full-screen disassemblies may be scrolled through with Pg up, Pg Dn, and arrow keys
- Blocks of disassembled code may be dumped to a disk file—blocks may be large, small, and/or discontinuous
- Single-step through full-screen disassemblies by hitting "+" key. Current instruction is underlined on display.
- Hundreds of simultaneous breakpoints supported. Groups of breakpoints may be tagged and toggled on/off by tag number.
- Individual breakpoints set by typing cntl-B on desired line of disassembly.
- Cntl-X starts up execution from underlined instruction.
- Saves user's graphic display when breakpoint hit, restores user's display when user's program started again. User's frozen display may be toggled to/from for observation when breakpoint hit. (Standard monochrome display subject to certain reasonable restrictions when using this feature.)
- F6 key will center disassembly display around line on which cursor is placed.
- User may type comments on disassembled code lines which will be retained throughout session
- Complete control to load your .EXE file almost anywhere in memory
- Dump display(s) may be brought up on separate windows (split-screens) or window-levels for continuous monitoring of selected memory areas. (Version 2.0 will allow user program to be running simultaneously while dump display is updating.)
- Alt-F10 key combination will interrupt program-under-test and bring up CodeSmith™ disassembly where break occurred
- Simple commands resemble Microsoft DEBUG commands, or use single-keystroke commands to speed your work
- Coded entirely in high-speed machine language
- Requires about 40K—multiple windows claim additional space on a dynamic basis
- Version 1.5 available April 1983—monochrome display version only (introductory price $145, updates $20)
- Version 2.0 (with color graphics compatibility) available July 1983 will include auto-assembly of user-entered patches, automatic label-generation option, symbolic debugging of certain PASCAL, C, and Assembler programs, 8087 disassembly and debug support, traceback of last several hundred instructions executed, watchpoint definitions (conditional halting of user program when data stored to certain locations, etc.)
- Version 3.0 available Fall 1983 will include full-screen editor option for updating your source file while still in CodeSmith™. MS-DOS Interface will be supported so that MS-DOS commands (compilations, etc.) may be entered from CodeSmith™.

CodeSmith™ has been designed with you in mind to substantially increase your ability to get your programs working—fast

VISUAL AGE
642 N. Larchmont Blvd., Los Angeles, CA 90004 (213) 464-8141

CodeSmith is a registered trademark of International Arrangements, Inc.
Microsoft and MS are registered trademarks of Microsoft Corp.
IBM is a registered trademark of International Business Machines Corp.

Circle **no. 86** on reader service card.

Augusta Part III

Listing Two

Source listing of the Augusta Compiler (Part 1 of 2 parts).

(Text begins on page 13)


```
1000 DEFINT A-Z:
       PRINT"Augusta(TM) Compiler V1.1A ":
       PRINT"(C) Copyright 1983 by Computer
       Linguistics":
       PRINT"All rights reserved."
1003 DIM MAP(26), KEY$(33), S$(100),
       TY(20), B$(3), B(3), S(500)
1010 PRINT:
       PRINT"Initializing":
       GOSUB 1780:
       GOSUB 1110:
       SI=1:
       INPUT"Source File ? ",S$:
       GOSUB 1230
1020 INPUT"Code File ? ",C$:
       OPEN"R",#1,C$,128:
       CLOSE 1:
       KILL C$
1040 INPUT"Listing(Y/CR)? ",T1$:
       OPEN"R",#1,C$,128:
       RO=16:
       MO=RO:
       IF T1$="Y" THEN PLST=-1:
       LPRINT LP$;
1050 GOSUB 1290:
       GOSUB 1400:
       PRINT FRE("");" Bytes for Symbols":
       GOSUB 1980
1060 PUT #1, RO:
       BAR=15:
       ID=16
1135 SC=17:
       COMMA=18:
       SEMICOLON=19:
       COLON=20:
       EQGT=21:
       COLONEQ=22:
       DOT=23:
       DOTDOT=24:
       CH=25:
       AT=26
1140 KAND=27:
       KARRAY=28:
       KBEGIN=29:
       KCASE=30:
       KCONST=31:
       KDECLARE=32:
       KELSE=33:
       KELSEIF=34:
       KEND=35:
       KEXIT=36:
       KFOR=37:
       KFUNC=38:
       KIF=39
1145 KIN=40:
       KIS=41:
       KLOOP=42:
       KLAST=43:
       KLEN=44:
       KMOD=45:
       KNOT=46:
       KNULL=47:
       KOF=48:
       KOR=49:
       KOTHERS=50:
       KOUT=51
1150 KPRAGMA=52:
       KPROC=53:
       KRET=54:
       KREVERSE=55:
       KTHEN=56:
       KWHEN=57:
       KWHILE=58
       PNOT=18:
       PADI=19:
       PNGI=20:
       PSBI=21:
       PMPI=22:
       PDVI=23:
       PIND=24:
       PEQUI=25:
       PNEQI=26:
       PLEQI=27:
       PSLDC=61:
       PINCL=80:
       PDECL=81
1210 PLESI=28:
       PGEQI=29:
       PGTRI=30:
       PEQUSTR=31:
       PNEQSTR=32:
       PLEQSTR=33:
       PLESSTR=34:
       PGEQSTR=35:
       PGTRSTR=36:
       PUJP=37:
       PFJP=38:
       PXJP=39


1215 PCLP=40:
       PCGP=41:
       PCSP=42:
       PRET=43:
       PMODI=45:
       PCIP=46:
       PRNP=47:
       PEOP=15:
       PSLDCN1=63:
       PIXA=48


1217 PSLDO=57:
       PSLAO=58:
       PSLLA=59:
       PSLDLO=49:
       PSLDL=60
1220 RETURN
```

```
        FIELD #1, 2 AS T1$, 2 AS T2$, 2 AS T3$,
        2 AS D$,2 AS S$
1070 LSET T1$=MKI$(GC):
        LSET T2$=MKI$(MO):
        LSET T3$=MKI$(PROC):
        LSET D$=MKI$(0):
        LSET S$=MKI$(1113)
1080 PUT #1,1:
        FIELD #1,128 AS D$:
        FOR I=1 TO MB:
        IF B(I)<>0 AND B(I)<>RO THEN LSET
        D$=B$(I):
        PUT #1, B(I)
1090 NEXT I:
        CLOSE 1:
        PRINT:
        PRINT "Compiled OK":
        PRINT LN;"lines. ";GC-1920;"Bytes":
        GOTO 32767



1110 'INIT
1120 QUOTE$=CHR$(34):
        LEXCH$=ALF$+DIG$+" @*+=-<>/:;')(,"+
        QUOTE$+".#!"+CHR$(3)+CHR$(96)+CHR$(9):
        CLST=-1
1130 SQUOTE=0:
        EOL=1:
        C=2:
        LP=3:
        RP=4:
        MUL=5:
        DIV=6:
        ADD=7:
        SUBT=8:
        LES=9:
        LEQ=10:
        GT=11:
        GEQ=12:
        EQ=13:
        NEQ=14:
```

(Continued on page 20, column 2)

```
1160 ADDOPS$=CHR$(ADD)+CHR$(SUBT):
        MULOPS$=CHR$(MUL)+CHR$(DIV)+CHR$(KMOD):
        LOGICALOPS$=CHR$(KAND)+CHR$(KOR)
1165 UNARYOPS$=CHR$(ADD)+CHR$(SUBT)+CHR$(KNOT):
        RELOPS$=CHR$(LES)+CHR$(LEQ)+CHR$(GT)+
        CHR$(GEQ)+CHR$(EQ)+CHR$(NEQ)
1170 DECLPARTS$=CHR$(ID)+CHR$(KPROC)+
        CHR$(KFUNC)+CHR$(KPRAGMA)
1180 STMTS$=CHR$(KWHILE)+CHR$(KFOR)+
        CHR$(KLOOP)+CHR$(KDECLARE)+CHR$(KBEGIN)+
        CHR$(KEXIT)+CHR$(KRET)+CHR$(KIF)
1185 STMTS$=STMTS$+CHR$(KCASE)+
        CHR$(KNULL)+CHR$(ID)+CHR$(KPRAGMA)
1190 LN=1:
        EOI=0:
        LL=0:
        CPROC=0:
        PROC=0:
        GC=1920:
        VLOC=VARPTR(V):
        VLOC1=VLOC+1:
        TSTR=0:
        TINT=1:
        TCHR=2:
        TBOL=4:
        FMSZ=14
1200 PLDCI=1:
        PLDL=2:
        PLLA=3:
        PLDB=4:
        PLDO=5:
        PLAO=6:
        PDUP=7:
        PLOD=8:
        PLDA=9:
        PPOP=10:
        PSTO=11:
        PSINDO=12:
        PLCA=13:
        PSAS=14:
        PAND=16
1205 POR=17:
```

(Continued on page 20, column 3)

```
1230 'OPEN SRC
1240 SI=SI+1:
        OPEN"I",#SI,S$:
        RETURN



1250 'LC TO UC
1260 IF INSTR(LC$,CH$) THEN CH$=CHR$(ASC(CH$)-32)
1270 RETURN



1280 'GETLINE
1290 LN=LN+1:
        IF EOF(SI) THEN CLOSE SI:
        SI=SI-1:
        IF SI>1 AND PLST THEN LPRINT
        TAB(26);"END OF INCLUDE"
1300 IF SI=1 THEN EOI=-1:
        RETURN
1310 LINE INPUT #SI, BUF$
1320 IF PLST=0 GOTO 1330 ELSE LPRINT
        USING "##### #### ###### ###### ";LN,
        CPROC,CP,OFST;:
        LPRINT BUF$
1325 IF (LN MOD 60)=0 THEN LPRINT CHR$(12);:
        LPRINT:
        LPRINT
1330 IF CLST<>0 THEN PRINT BUF$ ELSE IF
        (LN AND 63)=63 THEN PRINT LN;"..."
1340 IF LEN(BUF$)=0 GOTO 1290 ELSE
        BUF$=BUF$+CHR$(3):
        B=1:
        WHILE MID$(BUF$,B,1)=" ":
        B=B+1:
        WEND:
        CH$=MID$(BUF$,B,1):
        B=B+1:
        RETURN
```

(Continued on next page)

Augusta Part III
Listing Two

(Continued, text begins on page 13)

```
1360 'GETCH
1370 LSET CH$=MID$(BUF$,B,1):
       B=B+1:
       RETURN
1380 B=B+1
1390 RETURN

1400 'GETSYM
1410 OLDB=B:
       GOSUB 1250:
       I=INSTR(LEXCH$,CH$):
       IF I=0 THEN E=1:
       GOTO 5020
1420 IF I<27 THEN GOSUB 1460:
       GOTO 1430
1423 IF I<42 THEN ON I-26 GOSUB 1500,
       1500,1500,1500,1500,1500,1500,1500,1500,
       1500,1700,1770,1720,1520,1600:
       GOTO 1430
1427 ON I-41 GOSUB 1530,1620,1640,1660,
       1680,1540,1750,1570,1560,1550,1730,1580,
       1695,1710,1450,1440,1775
1430 IF EOI THEN E=12:
       GOTO 5020 ELSE IF OLDB=B GOTO 1410 ELSE
       LSET TT$=CHR$(T):
       RETURN
1440 T=SQUOTE:
       GOSUB 1360:
       RETURN
1450 GOSUB 1290:
       OLDB=B:
       RETURN
1460 S$="":
       WHILE INSTR(AN$,CH$):
       IF CH$<>"_" THEN S$=S$+CH$
1470 GOSUB 1370:
       GOSUB 1260:
       IF CH$="." THEN T=DOTDOT:
       GOSUB 1360
1590 RETURN
1600 GOSUB 1360:
       IF CH$=">" THEN T=EQGT:
       GOSUB 1360 ELSE T=EQ
1610 RETURN
1620 GOSUB 1360:
       IF CH$="=" THEN T=LEQ:
        GOSUB 1360 ELSE T=LES
1630 RETURN
1640 GOSUB 1360:
       IF CH$="=" THEN T=GEQ:
        GOSUB 1360 ELSE T=GT
1650 RETURN
1660 GOSUB 1360:
       IF CH$="=" THEN T=NEQ:
        GOSUB 1360 ELSE T=DIV
1670 RETURN
1680 GOSUB 1360:
       IF CH$="=" THEN T=COLONEQ:
       GOSUB 1360 ELSE T=COLON
1690 RETURN
1695 T=BAR:
       GOSUB 1360:
       RETURN
1700 WHILE CH$=" ":
       LSET CH$=MID$(BUF$,B,1):
       B=B+1:
       WEND:
       OLDB=B:
       RETURN
1710 T=BAR:
       GOSUB 1360:
       RETURN
1720 T=MUL:
       GOSUB 1360:
       RETURN
1730 I1=INSTR(B,BUF$,QUOTE$):
       IF I1=0 THEN E=10:
       GOTO 5020
1740 S$=MID$(BUF$,B,I1-B):
       T=SC:
       B=I1+1:
1795 D$=S$:
       LOC1=0:
       LOC2=0:
       V=0:
       VLOC1=0:
       VLOC=0:
       TN=0:
       TT$=S$:
       HASH=0:
       ID$=S$:
       BUF$=S$:
       T1=0:
       T2=0

1800 NKEY=33:
       SSP=1:
       MB=3:
       FOR I=0 TO MB:
       B$(I)=SPACE$(128):
       B(I)=0:
       NEXT I

1820 OPEN "I",#1,"KEYWORDS.TXT":
       LINE INPUT #1,LC$:
       T1=1:
       WHILE T1>0:
       INPUT #1,T1:
       LP$=LP$+CHR$(T1):
       WEND
1830 INPUT#1,DIG$,HDIG$,ALF$,LC$,AN$
1840 FOR I=1 TO 26:
       INPUT #1,MAP(I):
       NEXT I

1850 I=1:
       INPUT #1,ID$,TYPE,KIND,PINFO,CONST,
       OBJSZ,ADDR,LL:
       IF ID$<>"*END*" THEN ID$=ID$+SPACE$(8-
       LEN(ID$)):
       GOSUB 3850:
       GOTO 1850
```

```
        WEND:
        IF LEN(S$)>8 THEN S$=LEFT$(S$,8)
1480 ID$=S$+SPACE$(8-LEN(S$)):
        GOSUB 1890:
        RETURN
1490 'DIGITS
1500 TN=0:
        I1=10
1510 WHILE INSTR(HDIG$,CH$):
        TN=TN*I1+INSTR(HDIG$,CH$)-1:
        GOSUB 1360:
        WEND
1515 IF CH$="#" THEN I1=TN:
        TN=0:
        GOSUB 1360:
        GOTO 1510 ELSE T=C:
        RETURN
1520 T=ADD:
        GOSUB 1360:
        RETURN
1530 T=SUBT:
        GOSUB 1360:
        IF CH$="-" THEN GOSUB 1280:
        OLDB=B:
        RETURN ELSE RETURN
1540 T=SEMICOLON:
        GOSUB 1360:
        RETURN
1550 T=COMMA:
        GOSUB 1360:
        RETURN
1560 T=LP:
        GOSUB 1360:
        RETURN
1570 T=RP:
        GOSUB 1360:
        RETURN
1580 T=DOT:
        GOSUB 1360:
```

(Continued on page 22, column 2)

```
        GOSUB 1360:
        RETURN
1750 GOSUB 1360:
        GOSUB 1360:
        IF CH$<>"'" THEN E=11:
        GOTO 5020
1760 GOSUB 1360:
        GOSUB 1930:
        TN=ASC(MID$(S$,2,1)):
        T=CH:
        RETURN
1770 T=AT:
        GOSUB 1360:
        RETURN
1775 GOSUB 1360:
        OLDB=B:
        RETURN
1780 'READ DATA
1790 CH$=" ":
        B=0:
        LB=0:
        AN$=CH$:
        LC$=CH$:
        S$=CH$:
        T=0:
        T0=0:
        X=0:
        SP=0:
        TSP=0:
        LEXCH$=S$:
        CP=0:
        CB=0:
        W=0:
        I=0:
        R2=0:
        R1=0:
        T3=0:
        R0=16
```

(Continued on page 22, column 3)

```
1860 IF EOF(1) GOTO 1880 ELSE INPUT #1,T$:
        IF LEN(T$)>8 THEN T$=LEFT$(T$,8)
1870 T$=T$+SPACE$(8-LEN(T$)):
        KEY$(I)=T$:
        I=I+1:
        GOTO 1860
1880 CLOSE 1:
        KEY$(0)=" ":
        KEY$(NKEY)=" ":
        RETURN
1890 'LOOKUP KEYWORD
1900 HASH=MAP(INSTR(ALF$,LEFT$(ID$,1)))
1910 IF KEY$(HASH)=ID$ THEN T=HASH+26
        ELSE IF ASC(KEY$(HASH))<>ASC(ID$) THEN
        T=ID ELSE HASH=HASH+1:
        GOTO 1910
1920 RETURN
1930 'GET S$
1940 S$=MID$(BUF$,OLDB-1,B-OLDB):
        RETURN
1950 IF T0=T THEN RETURN
1955 E=4:
        GOSUB 5110:
        PRINT "REENTER+ ";:
        LINE INPUT T$:
        BUF$=LEFT$(BUF$,B-1)+T$+CHR$(3):
        GOSUB 1360:
        GOSUB 1400:
        GOTO 1950
1960 IF T0=T THEN GOSUB 1400:
        RETURN ELSE GOSUB 1950:
        GOSUB 1400:
        RETURN
```

(Continued on next page)

Augusta Part III
Listing Two

(Continued, text begins on page 13)

```
1970 'COMPILATION
1980 GOSUB 2770
1990 IF T=KPROC THEN GOSUB 1400:
       GOSUB 2010:
       TO=SEMICOLON:
       GOSUB 1950
2000 RETURN

2010 'PARSE PROC
2020 GOSUB 5200
2030 KIND=2:
       PROC=PROC+1:
       CPROC=PROC:
       ADDR=PROC:
       X=ADDR:
       GOSUB 4280:
       GOSUB 3850:
       GOSUB 1400
2040 OFST=-FMSZ:
       IF T=KIS GOTO 2060
2050 GOSUB 2100:
       TO=KIS:
       GOSUB 1950

2060 'IS
2070 X=-(OFST+FMSZ):
       GOSUB 4280:
       GOSUB 1400:
       OFST=0:
       MXOF=0:
       GOSUB 2440:
       W=PRET:
       GOSUB 3990:
       GOSUB 5300:
       RETURN
```

```
2235 T1$=MID$(T1$,9):
       OFST=OFST-2:
       WEND
2240 RETURN

2250 'SUBTYPEINDICATIONUNIT
2260 GOSUB 3890:
       IF KIND<>4 THEN E=8:
       GOTO 5020 ELSE IF PINFO=0 THEN KIND=1
       ELSE KIND=5
2280 IF TYPE<>0 THEN GOSUB 1400:
       RETURN

2285 GOSUB 2300:
       IF OBJSZ>255 THEN E=15:
       GOTO 5020 ELSE RETURN

2290 'GET C
2293 IF T<>ID THEN GOTO 2297 ELSE T8=TYPE:
       T3=KIND:
       T4=PINFO:
       T5=CONST:
       T6=OBJSZ:
       T7=LL

2294 GOSUB 3890:
       IF KIND=0 AND TYPE=1 THEN T=C:
       T2=CONST
2295 TYPE=T8:
       KIND=T3:
       PINFO=T4:
       CONST=T5:
       OBJSZ=T6:
       LL=T7
2297 TO=C:
       GOSUB 1960:
       RETURN
```

```
2400 TO=KIS:
       GOSUB 1960
2410 X=-(OFST+FMSZ):
       GOSUB 4280:
       OFST=0:
       MXOF=0:
       GOSUB 2440:
       GOSUB 5300:
       RETURN

2440 'BODYPART
2450 IF INSTR(DECLPARTS$,TT$) THEN GOSUB 2480
2460 CB=GC:
       CP=0:
       GOSUB 2790
2470 RETURN

2480 'DECLPART
2490 IF T=ID THEN T1$=ID$:
       K1=5:
       GOSUB 2560:
        GOTO 2540
2500 IF T=KPROC THEN GOSUB 1400:
        GOSUB 2010:
        GOTO 2540
2510 IF T=KFUNC THEN GOSUB 1400:
        GOSUB 2340:
        GOTO 2540
2520 IF T=KPRAGMA THEN GOSUB 2770:
       GOTO 2550

2530 E=3:
       GOTO 5020
2540 GOSUB 3420
2550 IF INSTR(DECLPARTS$,TT$) GOTO 2480
       ELSE GOSUB 4990:
       RETURN
```

```
2100 'PROCFORMALPART
2110 T2$="":
       T0=LP:
       GOSUB 1960
2120 GOSUB 2160:
       IF T=SEMICOLON THEN GOSUB 1400:
       GOTO 2120
2130 T0=RP:
       GOSUB 1960:
       FOR I=OFST TO -FMSZ-2 STEP 2:
       T1$=LEFT$(T2$,17):
       T2$=MID$(T2$,18):
       IF (LEN(S$(SSP))+17)>255 THEN SSP=SSP+1
2140 S$(SSP)=LEFT$(T1$,14)+MKI$(I)+
       RIGHT$(T1$,1)+S$(SSP):
       NEXT I
2150 RETURN
```

```
2160 'PROCPARAMDECL
2170 T1$=""
2180 T0=ID:
        GOSUB 1950:
       T1$=T1$+ID$:
       GOSUB 1400
2190 IF T=COMMA THEN GOSUB 1400:
        GOTO 2180
2200 T0=COLON:
       GOSUB 1960:
       P1=1:
       IF T=KOUT THEN P1=2:
       GOSUB 1400:
       GOTO 2220
2210 IF T=KIN THEN GOSUB 1400
2220 GOSUB 2250:
       PINFO=P1
2230 WHILE LEN(T1$)>0:
       T2$=T2$+LEFT$(T1$,8)+CHR$(TYPE)+
       CHR$(KIND)+CHR$(PINFO)+MKI$(CONST)+
       CHR$(OBJSZ)+MKI$(0)+CHR$(LL)
```

(Continued on page 24, column 2)

```
2300 'OBJSZ
2310 GOSUB 1400:
       IF T<>LP GOTO 2330 ELSE GOSUB 1400
2320 GOSUB 2290:
       OBJSZ=TN+1:
       T0=RP:
       GOSUB 1960
2330 RETURN
```

```
2340 'PARSEFUNC
2350 GOSUB 5200:
       KIND=3:
       PROC=PROC+1:
       CPROC=PROC:
       ADDR=PROC:
       X=ADDR:
       GOSUB 4280:
       GOSUB 3850:
       X=SSP:
       GOSUB 4280:
       X=LEN(S$(SSP))
2355 GOSUB 4280:
       GOSUB 1400
```

```
2370 OFST=-FMSZ:
       IF T=LP THEN GOSUB 2100
2380 T0=KRET:
       GOSUB 1960:
       GOSUB 2250:
       GOSUB 4300:
       T2=X:
       GOSUB 4300:
       T1=X:
       T3=LEN(S$(T1)):
       IF KIND<>5 OR OBJSZ<>2 THEN E=16:
       GOTO 5020
2385 S$(T1)=LEFT$(S$(T1),T3-T2+8)+
       CHR$(TYPE)+MID$(S$(T1),T3-T2+10)
```

(Continued on page 24, column 3)

```
2560 'OBJDECL
2570 GOSUB 1400
2580 IF T=COMMA THEN GOSUB 1400:
       T0=ID:
       GOSUB 1950:
       T1$=T1$+ID$:
        GOTO 2570
2590 T0=COLON:
       GOSUB 1960
```

```
2600 IF T=KCONST GOTO 2650
2610 IF T=KARRAY GOTO 2700
2620 GOSUB 2250:
       OBJSIZE=OBJSZ
2630 PINFO=0:
       KIND=K1:
       WHILE LEN(T1$)>0:
       ID$=LEFT$(T1$,8):
       T1$=MID$(T1$,9):
       ADDR=OFST:
       OFST=OFST+OBJSIZE:
       GOSUB 3850:
       WEND
2640 RETURN
```

```
2650 'CONSTANT
2670 K1=0:
       OBJSIZE=0:
       GOSUB 1400:
       T0=COLONEQ:
       GOSUB 1960:
       IF T=ID THEN GOSUB 3890:
       GOTO 2690 ELSE IF T=SUBT THEN T1=-1:
       GOSUB 1400 ELSE T1=1
2680 CONST=TN*T1:
       IF T=C THEN TYPE=1 ELSE TYPE=2
2690 GOSUB 1400:
       GOTO 2630
```

(Continued on next page)

Augusta Part III

Listing Two

(Continued, text begins on page 13)

```
2700 'ARRAY
2710 K1=1:
        GOSUB 1400:
        T0=LP:
        GOSUB 1960:
        T2=TN:
        GOSUB 2290:
        T0=RP:
        GOSUB 1960:
        T0=KOF:
        GOSUB 1960
2750 GOSUB 2250:
        CONST=T2:
        OBJSIZE=(T2+1)*OBJSZ:
        IF T2<0 OR T2>16383 THEN E=15:
        GOTO 5020 ELSE GOTO 2630


2770 'PRAGMA
2780 IF T<>KPRAGMA THEN RETURN ELSE GOSUB 4830:
        GOSUB 1280:
        GOSUB 1400:
        GOTO 2780


2790 'STMT
2800 T0=KBEGIN:
        GOSUB 1960:
        GOSUB 2810:
        T0=KEND:
        GOSUB 1960:
        RETURN


2810 'SEQOFSTMTS
2820 I=INSTR(STMTS$,TT$)
2825 IF I=0 THEN RETURN ELSE ON I GOSUB
        4320,4320,4320,2850,2850,2890,2930,2970,
        4630,2830,3440,2770:
        GOTO 2820
2930 'RETURN
2940 GOSUB 1400
2950 IF T<>SEMICOLON THEN GOSUB 3100:
        TSP=TSP-1:
        W=PRNP ELSE W=PRET
2960 GOSUB 3990:
        GOSUB 3420:
        RETURN


2970 'IF
2980 LUJP=0
2990 GOSUB 1400:
        GOSUB 3100:
        GOSUB 4930:
        W=PFJP:
        GOSUB 3990:
        X=CP:
        GOSUB 4280:
        GOSUB 4030:
        X=LUJP:
        GOSUB 4280
2995 T0=KTHEN:
        GOSUB 1960:
        GOSUB 2810:
        GOSUB 4300:
        LUJP=X


3000 IF T=KEND THEN GOSUB 3040:
        GOTO 3030
3010 IF T=KELSEIF THEN GOSUB 3060:
        GOSUB 3040:
        GOTO 2990
3020 T0=KELSE:
        GOSUB 1960:
        GOSUB 3060:
        GOSUB 3040:
        X=LUJP:
        GOSUB 4280:
        GOSUB 2810:
        GOSUB 4300:
        LUJP=X
3100 'EXPR
3110 GOSUB 3190:
        LFJP=0:
        PREV=0
3120 IF INSTR(LOGICALOPS$,TT$)=0 THEN
        IF PREV GOTO 3180 ELSE RETURN
3125 X=T:
        GOSUB 1400:
        IF (X=KAND AND T=KTHEN) THEN X=KAND+
        KTHEN ELSE IF (X=KOR AND T=KELSE) THEN
        X=KOR+KELSE
3130 IF PREV<>0 THEN IF PREV<>X THEN E=10:
        GOTO 5020
3140 IF X<>KAND AND X<>KOR GOTO 3160
3145 GOSUB 4280:
        GOSUB 3190:
        IF (TY(TSP)<>TBOL) OR (TY(TSP)<>TY(TSP-
        1)) THEN E=9:
        GOTO 5020
3147 TSP=TSP-1:
        GOSUB 4300:
        PREV=X:
        IF X=KAND THEN W=PAND ELSE W=POR
3150 GOSUB 3990:
        GOTO 3120
3160 GOSUB 4280:
        T1=X:
        W=PDUP:
        GOSUB 3990:
        IF T1=KAND+KTHEN THEN W=PFJP ELSE W=PNOT:
        GOSUB 3990:
        W=PFJP
3170 GOSUB 3990:
        W=LFJP:
        LFJP=CP:
        GOSUB 4030:
        GOSUB 1400:
        X=LFJP:
        GOSUB 4280:
        GOSUB 3190
```

(The Augusta Compiler listing will be continued in the July 1983 issue.)

```
2830 'NULL
2840 GOSUB 1400:
        GOSUB 3420:
        RETURN


2850 'BLOCK
2860 X=OFST:
        GOSUB 4280:
        OFST=OFST+2:
        GOSUB 5400:
        IF T=KDECLARE THEN GOSUB 1400:
        GOSUB 2480
2880 GOSUB 2790:
        GOSUB 5500:
        GOSUB 5700:
        GOSUB 4300:
        OFST=X:
        GOSUB 3420:
        RETURN



2890 'EXIT
2900 IF LPFLG=0 THEN E=14:
        GOTO 5020
2910 GOSUB 1400:
        IF T=SEMICOLON THEN W=PUJP:
        GOSUB 3990:
        GOTO 2925
2920 TO=KWHEN:
        GOSUB 1960:
        GOSUB 3100:
        GOSUB 4930:
        W=PNOT:
        GOSUB 3990:
        W=PFJP:
        GOSUB 3990
2925 W=XITJP:
        XITJP=CP:
        GOSUB 4030:
        GOSUB 3420:
        RETURN
```

(Continued on page 26, column 2)

```
3030 TO=KEND:
        GOSUB 1960:
        TO=KIF:
        GOSUB 1960:
        GOSUB 3080:
        GOSUB 3420:
        RETURN



3040 'FIX FJP
3050 GOSUB 4300:
        T1=CP:
        CP=X:
        W=T1-X-2:
        GOSUB 4030:
        CP=T1:
        RETURN



3060 'GEN UJP
3070 W=PUJP:
        GOSUB 3990:
        W=LUJP:
        LUJP=CP:
        GOSUB 4030:
        RETURN



3080 'FIXUP
3090 T2=CP:
        WHILE LUJP<>0:
        CP=LUJP:
        GOSUB 4010:
        LUJP=W:
        W=T2-CP-2:
        GOSUB 4030:
        WEND:
        CP=T2:
        RETURN
```

(Continued on page 26, column 3)

Augusta Part III
Listing Three
Simple expression parser.

```
PROCEDURE PARSEEXPR IS

PROCEDURE EXPR IS

  PROCEDURE SE IS

    PROCEDURE FACTOR IS
    BEGIN
      IF TOKEN='C' THEN
        GETTOKEN;            -- SCAN TO NEXT TOKEN
        RETURN;              -- RETURN TO CALLER
      END IF;
      IF TOKEN/='(' THEN ERROR ("SYNTAX ERROR");
      ELSE
        GETTOKEN;
        EXPR;
        IF TOKEN/=')' THEN ERROR (") EXPECTED");
        END IF;
      END IF;
    END; -- OF FACTOR

  BEGIN
    LOOP
      FACTOR;
      IF TOKEN='+' THEN
          NULL;              -- Do nothing for now
      ELSEIF TOKEN='-' THEN
          NULL;
      ELSE
          RETURN;
      END IF;
      GETTOKEN;
    END LOOP;
  END; -- OF SE

BEGIN
  LOOP
    SE;
    IF TOKEN='*' THEN
        NULL;
    ELSEIF TOKEN='/' THEN
        NULL;
    ELSE
        RETURN;
    END IF;
    GETTOKEN;
  END LOOP;
END;  -- OF EXPR

BEGIN
    GETTOKEN;
    EXPR;
END;
```

End Listing Three

(Listing Four begins on page 28)

Augusta Part III
Listing Four

Simple expression parser with code generation.

(Text begins on page 13)

```
PROCEDURE PARSEEXPR IS

PROCEDURE EXPR IS

  PROCEDURE SE IS

    PROCEDURE FACTOR IS
    BEGIN
      IF TOKEN='C' THEN
        EMITBYTE( 1 );     -- LDCI
        EMITWORD( TN );    -- the actual constant
        GETTOKEN;          -- SCAN TO NEXT TOKEN
        RETURN;            -- RETURN TO CALLER
      END IF;
      IF TOKEN/='(' THEN ERROR ("SYNTAX ERROR");
      ELSE
        GETTOKEN;
        EXPR;
        IF TOKEN/=')' THEN ERROR (")  EXPECTED");
        END IF;
      END IF;
    END; -- OF FACTOR


  BEGIN
    LOOP
      FACTOR;
      IF TOKEN='+' THEN
          EMITBYTE( + );          -- ADI
      ELSEIF TOKEN='-' THEN
          EMITBYTE( - );          -- SBI
      ELSE
          RETURN;
      END IF;
      GETTOKEN;
    END LOOP;
  END; -- OF SE


BEGIN
  LOOP
    SE;
    IF TOKEN='*' THEN
        EMITBYTE( * );           -- MPI
    ELSEIF TOKEN='/' THEN
        EMITBYTE( / );           -- DIVI
    ELSE
        RETURN;
```

```
(b)
     LLA        4          ; Load Address of I
     SLDC1                 ; Short load a constant 1
     STO                   ; I := 1;
!--> LDL        4          ; put I on stack
!    SLDC10                ; Short load 10 on stack
!    LEQI                  ; compare I <= 10
! (1)FJP        ---        ; Jump if I > 10
! !<------------<-
! !     .
! !     .
! !     .
! ! <code for enclosed sequence of statements>
! !     .
! !     .
! !     .
! !  INCL 4                ; Set I := I + 1
! !  UJP       ---         ; Jump back to comparison
!<--------------
  !
  !-->  <first instruction after the loop>
```

End Listing Five

Augusta Part III
Listing Six

(b) shows the code generated for the WHILE statement shown in (a).

```
(a)
     PROCEDURE DEMO IS
       I : INTEGER;

     BEGIN
       I:=1;
       WHILE I<10
       LOOP
         I:=I+1;
       END LOOP;
     END;

(b)

     SLAO       0          ; Load address of I
     SLDC1                 ; Short load constant 1
     STO                   ; Store, I:=1
  !->SLDO       0          ; Load value of I
  !  SLDC10                ; and compare to 10
  !  LESI                  ; Push -1 if >10
```

```
      END IF;
      GETTOKEN;
   END LOOP;
END;   -- OF EXPR



PROCEDURE STMT IS
BEGIN
   IF TOKEN/='I' THEN ERROR ("ID EXPECTED");
   ELSEIF TOKEN/='=' THEN ERROR("= EXPECTED");
   ELSE
      GETTOKEN;
      EXPR;
      IF TOKEN/=';' THEN ERROR ("; EXPECTED");
   END IF;
END;

BEGIN
   GETTOKEN;
   STMT;
END;
```

End Listing Four

Augusta Part III
Listing Five

Code generated for the FOR loop. At (a) is a sample FOR loop, and at (b) is the corresponding p-code translation.

```
(a)   FOR I IN 1..10
      LOOP
        .
        .
        .
        <sequence of statements>
        .
        .
        .
      END LOOP;
```

(Continued on page 28, column 2)

```
|<-- FJP      ----        ; Jump if I>10
| |   SLAO      0         ; Load address of I
| |   SLDO      0         ; Load value of I
| |   SLDC1               ; and add 1 to it
| |   ADI
| |   STO
| |<-UJP      ---         ; Jump back for more
|
|--->
```

End Listing Six

Augusta Part III
Listing Seven

Code generation for an IF-THEN-ELSE. (a) shows the IF-THEN-ELSE statement and (b) shows the corresponding compiled code.

```
(a)
      IF A=B THEN
         C=A*2;
      ELSE
         C=B+A;
      END IF;

(b)
      LDL        0       ; Load value of A
      LDL        2       ; Load value of B
      EQUI               ; Pop top 2 words and compare, push TRUE/FALSE
|<--  FJP        --      ; Jump if they are not equal to do ELSE part
|     LLA        4       ; Load address of C
|     LDL        0       ; Load value of A
|     SLDC2              ; Load constant 2 to multiply by
|     MPI                ; Multiply A*2
|     STO                ; and set C:=A*2
|  |- UJP        --      ; jump around the ELSE part
|  |                     ; Following code is ELSE part
|--> LLA         4       ; Load address of C
   |  LDL        2       ; Load value of B
   |  LDL        0       ; And load value of A
   |  ADI                ; Add them together
   |  STO                ; and store result
   |-->
```

End Listing Seven

(Listing Eight begins on page 30)

Augusta Part III
Listing Eight

"Short-circuit" conditions are provided to evaluate only a portion of an expression. For example, the second expression (J/N > 50), will only be evaluated if the first part tests as true. Augusta provides two short circuit operators — AND THEN and OR ELSE.

(a)

```
    IF (N /= 0) AND THEN ( J / N > 50) THEN I:=0;
```

(b)

```
      SLDO     8         ; Get value of N
      SLDCO              ; load constant 0
      NEQI               ; see if not equal
      DUP                ; duplicate top of stack value
  |<--FJP      ----      ; Jump around if equal
  |   SLDO     10        ; Load J
  |   SLDO     8         ; Load N
  |   DVI                ; J / N
  |   SLDC     50        ; Load 50
  |   GTRI               ; > Comparison
  |   AND                ; AND with previous expression
|-|-->FJP      ----      ; and jump if false
|     SLAO     0         ; Load address of I
|     SLDCO              ; Load constant 0
|     STO                ; and store result
|
|-->
```

End Listing Eight

Augusta Part III
Listing Nine

(b) shows the code generated for the CASE statement shown in (a). The text describes the XJP instruction in greater detail.

(a)

```
    CASE I OF
      WHEN 7  =>  <statements 1>
      WHEN 11 =>  <statements 2>
      WHEN 19 =>  <statements 3>
      WHEN 23 =>  <statements 4>
    END CASE;
```

(b)

```
      LDL       2             ; Load value of I
      XJP    ___, ___, ___    ; Jump indirectly, using top of stack
              w1   w2   w3    ; value,-w1, as an index in to the
                        |     ; jump table at offset w3
                        |     ;
|<-----------------------
|
| |----> <code for statements 1>
| |                           ; This is case 7
| |      UJP ___              ; Jump past the jump table
| |
| | |--> <code for statements 2>
| | |                         ; This is case 11
| | |    UJP ___              ; Jump past the jump table
| | |
| | |    <code for statements 3>
| | |                         ; This is case 19
| | |    UJP ___              ; Jump past the jump table
| | |
| | |    <code for statements 4>
| | |                         ; This is case 23
| | |    UJP ___              ; Jump past the jump table
| | |
| | |                         ; The jump table follows
|------> UJP <-----|
  | |    ___ -->|  |          ; address of OTHERS condition
  | |           |  |
  |<---- ___    |  |          ; address of case 7
    |           |  |
    |    ___ ------>|         ; address of case 8 (pointer to OTHERS)
    |           |  |
    |    ___ ------>|         ; address of case 9 (pointer to OTHERS)
    |           |
    |    ___ ------>|         ; address of case 10 (to OTHERS)
    |           |
    |<-- ___    |             ; address of case 11
                              .
                              .
                              .
                |
         ___    |             ; address of case 19
                |
                              .
                              .
                              .
                |
         ___    |             ; address of case 23
                |
   <----------- |             ; first statement following END CASE
```

End Listing Nine

A Fast Circle Routine

In some instances it is necessary to draw partial circles or arcs and at other times complete circles are required. The algorithm presented here draws complete circles accurately and quickly and is relatively simple to program. The routine is written for the IBM Personal Computer using IBM's Macro Assembler, and can be implemented as an IBM Pascal procedure.

The Algorithm

Circle algorithms generally use a transformation from polar to rectangular coordinates by the following relationships:

by Daniel L. Lee

Daniel L. Lee, 1401 E. 55th Street, Apt. 601, Chicago, IL 60615.

$$X = R \text{ cosine}(A)$$
$$Y = R \text{ sine}(A)$$

where the (X,Y) coordinates produce the locus of points for the given radius R as the angle A varies from 0 to 2 pi radians. This method requires trigonometric routines to compute sine and cosine values at least for the start and end points and for an angular increment.

A simpler method which does not require sine and cosine values is obtained by elementary calculus, which can be used to show that dX/dY=-tangent(A)=-Y/X. Also, dY/dX=-cotangent(A)=-X/Y. In other words, the change in X, given a small change in Y, is simply the negative of the ratio of Y to X. Similarly, the change in Y, given a small change in X, is simply the negative of the ratio of X to Y.

Given these relationships, the circle procedure is straightforward. For simplicity it will be assumed that the circle center is at the origin of our coordinate system. It is not difficult to adjust for a change in origin.

The first point plotted is chosen as Y=R and X=O, where R is the radius measured in screen column units. Y is incremented by one unit and X is decremented by Y/X units and the point plotted. This process is continued until Y/X=1. At that point (which corresponds to a 45 degree angle), we switch over to decreasing X by one unit and increasing Y by X/Y units. The process is continued until X/Y equals 0, i.e., when X is decremented to 0. At that point an arc from 0 to 90 degrees has been drawn. The complete circle can be obtained by using the symmetry of the circle to compute the corresponding points in each of the three remaining circle quadrants as we trace out the arc from 0 to 90 degrees.

The switch over when Y/X equals 1 is required because the tangent will approach infinity as the angle approaches 90 degrees, and overflow will occur somewhere prior to that point.

To actually implement the algorithm, we have to adjust for the origin located at the top left-hand corner of the video screen. Also we have to take into account the aspect ratio of the screen. When plotting 0 to 45 degrees, the X value must be multiplied by the inverse of the aspect ratio, and when plotting from 45 to 90 degrees, the Y value must be multiplied by the aspect ratio. Assuming a screen aspect ratio of 4/3, the aspect ratio to produce a circle in medium-resolution graphics is 5/6 and is 5/12 for high-resolution graphics.

The Assembly Routine

The routine incorporates the elements of the algorithm just described. It is also necessary to scale the computations by some factor to retain numerical accuracy, and then to rescale to obtain the points to plot. A factor of 1000 gives sufficient accuracy for the PC's high-resolution mode.

As mentioned, the circle routine can be implemented as a Pascal procedure. Listing 1 (page 35) is the circle routine and Listing 2 (page 37) is a simple calling routine which generates an internal calling sequence as used by IBM Pascal.

Pascal sets up a frame on the stack for an active procedure which is invoked by a FAR call. First, the calling program pushes the procedure arguments from left to right onto the stack. Then the caller's segment address and offset are pushed onto the stack. When the procedure receives control, it should first save the caller's base pointer by pushing BP onto the stack. Then the BP register is set equal to the SP register (stack pointer) and the procedure can then reference the arguments by adding six bytes to the BP register – four for the caller's return address and two for the caller's base pointer which has been pushed onto the stack. Thus, assuming integer value arguments, the first variable on the stack can be addressed by BP+6, the second by BP+8, and so on.

To return, the procedure must restore the caller's base pointer in BP and the SP register must be restored by a RET x, where x is the number of bytes on the stack occupied by the arguments which were passed.

Note that the IBM-supplied BIOS routine is used to write the points to screen. That is accomplished by storing 12 in the AH register, the X coordinate in the CX register, the Y coordinate in the DX register, and then issuing an INT 10H instruction. In this routine, the INT 10H instruction is defined as the MACRO, BIOSCALL, the expansion of which is denoted by "+" in the listing.

DDJ

(Listing begins on page 35)

C compilers and Cross compilers

TELECON'S C COMPILERS OFFER YOU

- FULL **C**
- UNIX* Ver. 7 COMPATABILITY
- NO ROYALTIES ON GENERATED CODE
- GENERATED CODE IS REENTRANT
- **C** AND ASSEMBLY SOURCE MAY BE INTERMIXED
- UPGRADES & SUPPORT FOR 1 YEAR

IN THESE CONFIGURATIONS:

HOST	6809 TARGET	PDP-11*/LSI-11* TARGET	8080/(Z80) TARGET	8088/8086 TARGET
FLEX*/UNIFLEX* **OS-9***	$200.00 WITHOUT FLOAT $350.00 WITH FLOAT	500.00	500.00	500.00
RT-11*/RSX-11* PDP-11*	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT	500.00	500.00
CP/M* 8080/(Z80)	500.00	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT	500.00
PCDOS*/CP/M86* 8088/8086	500.00	500.00	500.00	200.00 WITHOUT FLOAT 350.00 WITH FLOAT

Others Pending

C SOURCE AVAILABLE FOR $2,500[00]

SO . . . IF YOU'RE READY TO MOVE UP TO **C** ...

CALL

408-275-1659

TELECON SYSTEMS
1155 Meridian Avenue, Suite 218
San Jose, California 95125

*PCDOS is a trademark of IBM CORP. MSDOS is a trademark of MICROSOFT. UNIX is a trademark of BELL LABS. RT-11/RSX-11/PDP-11 is a trademark of Digital Equipment Corporation. FLEX/UNIFLEX is a trademark of Technical Systems consultants. CP/M & CP/M86 are trademarks of Digital Research. OS-9 is a trademark of Microware & Motorola.

Circle no. **81** on reader service card.

Fast Circle Plot (Text begins on page 32)

Listing One

```
                              TITLE 'Fast Circle Plot'

                              BIOSCALL MACRO
                                    INT 10H       ;BIOS service id in AH
                                    ENDM

0000                          STACK SEGMENT PARA STACK 'STACK'
0000      40 [                     DB 64 DUP('STACK   ')
              53 54 41 43
              4B 20 20 20
                         ]

0200                          STACK ENDS

0000                          CSEG  SEGMENT PARA 'CODE'
                              ;-------------------------------------------
                              ;PROCEDURE  CIRCLE(X,Y,RADIUS,NUMER,DENOM,
                              ;                          COLOR:INTEGER)
                              ;Dan Lee  July 1, 1982
                              ;SourceWare
                              ;
                              ;draws a circle at center (x,y) with aspect
                              ;ratio numer/denom; radius in column units
                              ;
                              ;assumes entry via inter-segment call
                              ;
                              ;FRAME:   VALUE X      : BP+16
                              ;         VALUE Y      : BP+14
                              ;         VALUE RADIUS: BP+12
                              ;         VALUE NUMER : BP+10
                              ;         VALUE DENOM : BP+8
                              ;         VALUE COLOR : BP+6
                              ;-------------------------------------------
0000                          CIRCLE  PROC FAR
                                      ASSUME CS:CSEG,SS:STACK
                                      PUBLIC CIRCLE
0000  55                              PUSH BP          ;caller's frame pntr
0001  8B EC                           MOV BP,SP        ;set proc frame pntr
0003  8B 46 0A                        MOV AX,[BP+10] ;get aspect numer and
0006  BB 03E8                         MOV BX,1000      ;scale it by 1000
0009  F7 EB                           IMUL BX
000B  8B 4E 08                        MOV CX,[BP+8]  ;get aspect denom
000E  F7 F9                           IDIV CX          ;AX=aspect*1000
0010  50                              PUSH AX          ;store aspect*1000
0011  91                              XCHG AX,CX       ;get denom in AX
0012  8B 4E 0A                        MOV CX,[BP+10] ;get numer in CX
0015  F7 EB                           IMUL BX          ;AX=denom*1000
0017  F7 F9                           IDIV CX          ;AX=inv aspect*1000
0019  89 46 08                        MOV [BP+8],AX  ;store it
001C  58                              POP AX           ;get aspect*1000
001D  89 46 0A                        MOV [BP+10],AX ;and store it
                              ;
                              ; start by incrementing Y by one unit and
                              ; decrementing X by TAN units*inv aspect
                              ; start at (RADIUS,Y) and plot to 45 deg
                              ;
0020  8B 46 0C                        MOV AX,[BP+12] ;get radius for
0023  BB 03E8                         MOV BX,1000      ;initial X and scale
0026  F7 EB                           IMUL BX          ;up by 1000
0028  33 FF                           XOR DI,DI        ;zero initial Y value

002A  50                      CR5:    PUSH AX          ;save lo word X*1000
002B  52                              PUSH DX          ;save hi word X*1000
002C  33 DB                           XOR BX,BX        ;begin round process
002E  05 01F4                         ADD AX,500       ;'one-half'
0031  13 D3                           ADC DX,BX
0033  BB 03E8                         MOV BX,1000      ;rescale X by 1000
0036  F7 FB                           IDIV BX          ;to graph
0038  8B D8                           MOV BX,AX        ;BX=1st quad X
003A  03 46 10                        ADD AX,[BP+16] ;add X origin
003D  8B 56 0E                        MOV DX,[BP+14] ;get Y origin
0040  2B D7                           SUB DX,DI        ;and sub Y to plot
0042  8B C8                           MOV CX,AX        ;get X to plot
0044  8A 46 06                        MOV AL,[BP+6]  ;get color
0047  B4 0C                           MOV AH,12        ;write dot funct select
0049  50                              PUSH AX          ;save write dot parms
                                      BIOSCALL         ;write 1st quad point
004A  CD 10                   +       INT 10H          ;BIOS service id in AH
004C  58                              POP AX           ;restore write dot parms
```

(Continued on next page)

Fast Circle Plot

Listing One

(Listing One continued, text begins on page 32)

```
004D  2B CB                  SUB CX,BX      ;get 2nd quad
004F  2B CB                  SUB CX,BX      ;X+origin
0051  50                     PUSH AX        ;save write dot parms
                             BIOSCALL       ;write 2nd quad point
0052  CD 10            +     INT 10H        ;BIOS service id in AH
0054  58                     POP AX         ;restore write dot parms
0055  03 D7                  ADD DX,DI      ;get 3rd quad
0057  03 D7                  ADD DX,DI      ;Y+origin
0059  50                     PUSH AX        ;save write dot parms
                             BIOSCALL       ;plot 3rd quad point
005A  CD 10            +     INT 10H        ;BIOS service id in AH
005C  58                     POP AX         ;restore write dot parms
005D  03 CB                  ADD CX,BX      ;get 4th quad
005F  03 CB                  ADD CX,BX      ;X+origin
                             BIOSCALL       ;plot 4th quad point
0061  CD 10            +     INT 10H        ;BIOS service id in AH
                        ;
                        ; cx now at original point
                        ;
0063  87 CB                  XCHG CX,BX     ;get 1st quad X
0065  47                     INC DI         ;get new Y
0066  8B C7                  MOV AX,DI      ;AX=Y
0068  8B 5E 08               MOV BX,[BP+8]  ;BX=inv aspect*1000
006B  F7 EB                  IMUL BX        ;AX=Y*inv aspect*1000
006D  F7 F9                  IDIV CX        ;AX=TAN*inv aspect*1000
006F  33 D2                  XOR DX,DX      ;zero remainder
0071  8B F0                  MOV SI,AX      ;SI=TAN*inv aspect*1000
0073  F7 FB                  IDIV BX        ;AX=TAN
0075  3D 0001                CMP AX,1       ;TAN=1?
0078  5A                     POP DX         ;DX=hi word X*1000
0079  58                     POP AX         ;AX=lo word X*1000
007A  73 0B                  JAE CR7        ;yes, go to next sector
007C  F7 DE                  NEG SI         ;to decrement *X
007E  BB FFFF                MOV BX,-1      ;negative carry
0081  03 C6                  ADD AX,SI      ;new X value
0083  13 D3                  ADC DX,BX      ;hi word carry
0085  EB A3                  JMP SHORT CR5  ;plot new point
00C3  58                     POP AX         ;restore write dot parms
00C4  03 CF                  ADD CX,DI      ;get 4th quad
00C6  03 CF                  ADD CX,DI      ;X to plot
                             BIOSCALL       ;plot 4th quad point
00C8  CD 10            +     INT 10H        ;BIOS service id in AH
00CA  2B 56 0E               SUB DX,[BP+14] ;DX=Y-Y origin
00CD  F7 DA                  NEG DX         ;Y origin adjust
00CF  87 CA                  XCHG CX,DX     ;CX=Y
00D1  0B FF                  OR DI,DI       ;90 deg?
00D3  78 1D                  JS CR11        ;yes,exit
00D5  4F                     DEC DI         ;get new X
00D6  8B C7                  MOV AX,DI      ;AX=X
00D8  8B 5E 0A               MOV BX,[BP+10] ;BX=aspect*1000
00DB  F7 EB                  IMUL BX        ;AX=aspect*1000*X
00DD  F7 F9                  IDIV CX        ;AX=aspect*1000*COT
00DF  8B F0                  MOV SI,AX      ;SI=change in Y

00E1  5A                     POP DX         ;DX=hi word Y*1000
00E2  58                     POP AX         ;AX=lo word Y*1000
00E3  33 DB                  XOR BX,BX
00E5  0B F6                  OR SI,SI       ;for sign check
00E7  79 03                  JNS CR10       ;positive
00E9  BB FFFF                MOV BX,-1      ;negative carry
00EC  03 C6            CR10: ADD AX,SI      ;AX=new X value
00EE  13 D3                  ADC DX,BX      ;hi word carry
00F0  EB 9F                  JMP SHORT CR8  ;plot next point
                       ;
                       ; exit
                       ;
00F2  83 C4 04         CR11: ADD SP,4       ;adjust stack pntr
00F5  5D                     POP BP         ;caller's frame pntr
00F6  CA 000C                RET 12         ;release parms
00F9                   CIRCLE ENDP
00F9                   CSEG   ENDS
                              END
```

End Listing One

```
                            ;
                            ; plot 45 to 90 degrees
                            ; now decrease X by one unit and
                            ; increase Y by COT units*aspect ratio
                            ;
0087  8B C7                 CR7:   MOV AX,DI       ;get next Y to plot and
0089  BB 03E8                      MOV BX,1000     ;scale by 1000
008C  F7 EB                        IMUL BX         ;DX:AX=Y*1000
008E  8B F9                        MOV DI,CX       ;DI=last X value
0090  4F                           DEC DI          ;next X to plot

0091  50                    CR8:   PUSH AX         ;save lo word Y*1000
0092  52                           PUSH DX         ;save hi word Y*1000
0093  33 DB                        XOR BX,BX       ;begin round process
0095  05 01F4                      ADD AX,500      ;'one-half'
0098  13 D3                        ADC DX,BX
009A  BB 03E8                      MOV BX,1000     ;rescale Y to plot
009D  F7 FB                        IDIV BX         ;AX=Y
009F  8B D8                        MOV BX,AX       ;BX=1st quad Y coord
00A1  03 46 0E                     ADD AX,[BP+14]  ;add Y origin
00A4  8B 4E 10                     MOV CX,[BP+16]  ;CX=X origin
00A7  03 CF                        ADD CX,DI       ;X to plot
00A9  8B D0                        MOV DX,AX       ;Y to plot
00AB  8A 46 06                     MOV AL,[BP+6]   ;get color
00AE  B4 0C                        MOV AH,12       ;write dot funct select
00B0  50                           PUSH AX         ;save write dot parms
                                   BIOSCALL        ;write 1st quad point
00B1  CD 10                 +      INT 10H         ;BIOS service id in AH
00B3  58                           POP AX          ;restore write dot parms

00B4  2B CF                        SUB CX,DI       ;get 2nd quad
00B6  2B CF                        SUB CX,DI       ;X to plot
00B8  50                           PUSH AX         ;save write dot parms
                                   BIOSCALL        ;plot 2nd quad point
00B9  CD 10                 +      INT 10H         ;BIOS service id in AH
00BB  58                           POP AX          ;restore write dot parms
00BC  2B D3                        SUB DX,BX       ;get 3rd quad
00BE  2B D3                        SUB DX,BX       ;Y to plot
00C0  50                           PUSH AX         ;save write dot parms
                                   BIOSCALL        ;write 3rd quad point
00C1  CD 10                 +      INT 10H         ;BIOS service id in AH
```

(Continued on page 36, column 2)

Listing Two

```
                                   TITLE CALLER - CALLS CIRCLE SUBROUTINE

                                         EXTRN CIRCLE:FAR;external subroutine

0000                               STACK SEGMENT PARA STACK 'STACK'
0000      40 [                           DB  64 DUP('STACK   ')
                53 54 41 43
                4B 20 20 20
                            ]

0200                               STACK ENDS

0000                               CALLER SEGMENT PARA 'CODE'
0000                               START PROC FAR
                                         ASSUME CS:CALLER,SS:STACK
0000  1E                                 PUSH DS      ;save ret seg on stack
0001  33 C0                              XOR AX,AX    ;zero for ret offset
0003  50                                 PUSH AX      ;and save on stack
0004  B0 04                              MOV AL,4     ;color/graphics select
0006  CD 10                              INT 10H      ;set mode
0008  B8 00A0                            MOV AX,160   ;X origin coord
000B  50                                 PUSH AX      ;save on stack
000C  B8 0064                            MOV AX,100   ;Y origin coord
000F  50                                 PUSH AX      ;save on stack
0010  B8 0032                            MOV AX,50    ;circle radius
0013  50                                 PUSH AX      ;save on stack
0014  B8 0005                            MOV AX,5     ;aspect numer
0017  50                                 PUSH AX      ;save on stack
0018  B8 0006                            MOV AX,6     ;aspec denom
001B  50                                 PUSH AX      ;save on stack
001C  B8 0001                            MOV AX,1     ;foregrnd color
001F  50                                 PUSH AX      ;save on stack
0020  9A 0000 ---- E                     CALL CIRCLE;draw circle
0025  CB                                 RET          ;far ret to DOS
0026                               START ENDP
0026                               CALLER ENDS
                                         END
```

End Listing Two

Enhancing the C Screen Editor

In the January 1982 issue of Dr. Dobb's *we presented Edward Ream's popular Small-C screen editor. Readers received a fine editor which, at the same time, was an excellent candidate for extension and modification. Alan Howard has provided us with an extensive set of enhancements. Those who wish to implement changes in the original editor should find guidance and inspiration from Mr. Howard's work.*

Mr. Howard has chosen a modular approach to the problem and his changes should be easy to implement. Mr. Ream himself has attacked the situation by significant revision of the entire editor. We will be publishing Mr. Ream's revision, called RED, in the near future. Comparison of these two approaches should provide insight into programming techniques and implementation tradeoffs, and we feel fortunate to be able to publish both pieces of work.

While the January and May 1982 back issues are no longer available (the bound volume containing them should be out later this year), we understand the original editor may still be obtained from a variety of sources, including several computer bulletin boards. Edward Ream still distributes it himself for $50 on 8-inch, CP/M diskettes, and can be reached at 1850 Summit Avenue, Madison, WI 53705. A BDS C version may be obtained from The C User's Group, Box 287, Yates Center, KS 66783 (which sells it for $8 in a variety of formats) and Steve Passe's Cnode computer bulletin board (contact the C Users' group for details).

At the beginning of Edward Ream's article introducing his Small-C Screen Editor in *Dr. Dobb's Journal* (No. 63, January 1982), he asked whether our present editor lacked flexibility, transportability, and extensibility. Although my mainstay editor has been PIE from Software Toolworks, I have often been annoyed at its inability to edit large files or to make major structural changes in text. On the other hand, Ream's editor as presented did not go much beyond PIE and seemed clumsy in its Edit mode. It did, however, encourage adapting it to fit individual needs since it was presented in source code. Thus, after several nights of typing and several more nights of debugging, I had the C editor up and running (those of you who want to join me doing it the hard way, be sure to note the bug fixes in the May issue, No. 67). I have been modifying the editor during the last few months and it has evolved to the point that I feel some of *Dr. Dobb's* readers may be interested. The changes are of several types:

by Alan D. Howard

Alan D. Howard, Route 3, Box 680, Crozet, VA 22932.

(1) Extensions to the file-handling and buffer management to permit editing large files, extracting parts of the text to the write file, and moving and copying portions of the text within the file.

(2) Changing the edit mode operation to recognize special keys on the H19 Terminal or H89 Computer instead of single-letter commands. As a bonus, the editor now assumes *eXchange* mode for any other typed key. The code can be easily modified for other terminal protocols.

(3) Addition of a few new commands to the edit-mode and some changes in rules of operation.

(4) Slight modifications to allow modular compilation using the Software Toolworks C/80 compiler.

(5) Enhancements to improve the speed of response in edit mode and during search and replace operations.

These enhancements are discussed below as explanations for the listing that accompanies this article. Only those variables, declarations, and functions that have been changed from the original listing are listed. A documentation file is included in the listing which summarizes the expanded features and all commands.

Changes to File Handling and Command-Mode Operation

Most of the changes to the command-mode are concerned with adding flexibility to file handling and major buffer alterations. Separate read and write files are now maintained, with *load* <filename> specifying the read file name, clearing the buffer, reading the file, and closing the read file if it all fits in the buffer. Otherwise as much as fits is read in, and the read file remains open. If a filename is specified in the command line when the editor is loaded, the filename is automatically passed to the *load* command. The *open* <filename> command opens the read file without clearing the buffer or reading anything in. More from the read file can be added to the buffer with either the *rest* or *read* <n> commands. *Rest* reads in as much more as possible and gives the option of clearing the buffer, but *read* <n> reads only <n> more lines without clearing the buffer. The write file is specified by *name* <filename> for a new file, or *delname* <filename> for writing over an existing file (this replaces the *resave* command). *Rename* <filename> closes the existing write file and opens a new one. *Write* <n> writes <n> lines from the buffer to the write file, deleting those lines. To balance the existing *append* <filename> command, an *extract* <from to> command has been added to write the indicated line range to the write file without deleting those lines from the buffer. Finally, to allow flexibility, the read and write files can be closed by *closeread* and *closewrite*.

Major structural changes in the file can now be made using the *copy* <from to n> and *move* <from to n> commands, which take n lines from <from> and copy them to before line <to>. In addition, *move* deletes the <n> lines at <from>. For speed, these routines open up the new space, and then do the copying or moving. No move or copy is allowed if there is not room in the buffer. If buffer space is tight, or if the move or copy is to a part of the file not currently in memory, then use the *extract* and *append* commands, which use the disk as a buffer.

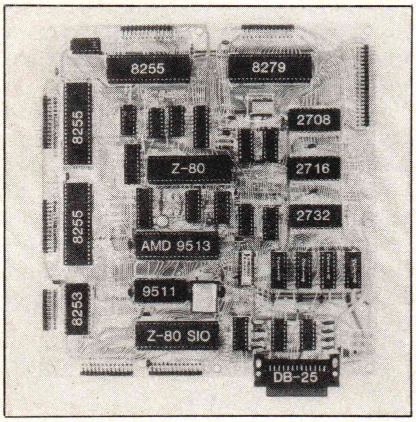
Because tabs are treated as ordinary characters, *showtab* has been included to allow tabs to be shown in reverse video. This mode is reset by *hidetab* (the default mode) where tabs are just blanks.

The following command-mode commands are unaltered from the original: *append, change, clear, delete, dos, find, g, list, search,* and *tabs.*

Changes to Edit and Insert Modes

The most important change to the edit mode is the replacement of the single-letter commands by escape-character sequences generated by the special keys on the H19/H89 keyboard. Similar keys, but with different escape sequences, are found on many other terminals and computers. This allows default use of the *eXchange* mode (replacing contents of cursor position with the typed keystroke) for normal keys, including the space bar. The *eXchange* mode has been speeded up considerably to allow rapid typing. Most special characters and edit mode commands retain their original functions (see the documentation portion of the listing

FREE BASIC Z-80 BOARD COMPUTER



The MASTER CONTROLLER BOARD contains:
-Z-80 Microprocessor
72-Parallel I/O lines; three 8255s
-Keyboard controller: 8279
12K-EPROM: three sockets for 2708, 2716, 2732
2K-RAM: 2114s
8-Sixteen bit counter timer channels: one 8253 and one AMD 9513
2-Serial I/O ports; one Z-80 SIO chip. One port is RS-232 W/DB-25
1-High speed arithmetic processor: AMD 9511
A bus expansion connector is provided
All this on one board less than nine inches on a side

Bare Controller Board with Doc. **$49.95**
Free Controller Basic is a public domain Tiny Basic that can IN and OUT ports, PEAK and POKE RAM, CALL assembly language programs, and use either DECIMAL OR HEXIDECIMAL numbers. In a 2716. Requires 2k RAM,SIO,8253 (baud gen.) With the BARE BOARD **$14.95** Alone **$19.95**
TDL monitor program allows a CRT or TTY to control the MASTER CONTROLLER BOARD. Requires 2k RAM,SIO,8253 (baud gen.), 4Mhz XTAL. Includes Complete Listing on a 2732 **$69.95**
Assembled TINY BASIC CONTROLLER BOARD has 2k RAM,SIO,8253 (baud gen.), 8255. This arrangement gives 24 I/O lines, 2 spare counter timer channels, and a serial channel available after using one counter timer channel as a baud gen. and one serial channel to talk to a terminal or computer. Functions can be expanded by adding additional RAM/ROM, I/O and processing chips. EXPANDABLE SPECIAL **$299.99**

OEM & Dealer Inquiries Welcome
USA & CANADA include $4.95 postage & handling. We ship World Round. Please include 20% for shipping plus $5 handling we refund the excess.

SPACE-TIME PRODUCTIONS
2053 N. Sheffield
Chicago, Illinois 60614
(312) 327-0391

Circle **no. 79** on reader service card.

Introducing SPL the first multi-mode spooler for CP/M® computers

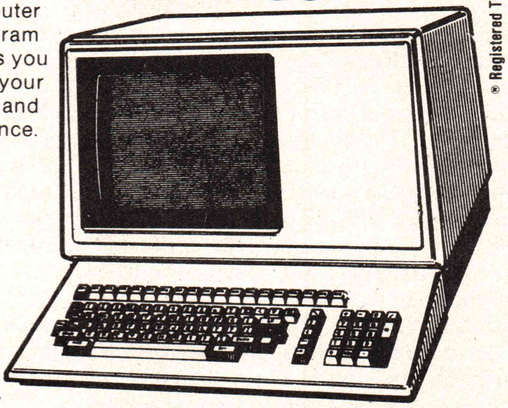
If you believed that your computer couldn't do better than a single task system think again. You can convert your machine into a dual-task computer with **SPL**, the amazing Spooler program developed by Blat R+D. **SPL** enables you to use hidden capacity available on your CP/M computer to print documents and run your ordinary programs, all at once.

While printing, your regular programs won't stop processing, waiting for the printer to finish. **SPL** will store the information to be printed in internal or external (disk drives) memory until the printer is ready to receive the data. Result: your programs will run at full speed.

As **SPL** can use up to the full capacity of your disks for temporary storage, it's much more powerful than hardware spoolers, which are limited to 64k memory or less.

SPL is an advanced product with several modes of operation. In addition to intercepting the output to the printer, **SPL** can print your existing text files, or those that your programs will create from now on. **SPL** will even take care of tab expansion. As an added bonus, **SPL** needs no installation on most CP/M 2.x computers.

If you have a computer
Get A Second Computer FOR $139



® Registered Trademark of Digital Research

You could get an equivalent increase in computing power by spending $1000 to $3000, but **SPL** is only $ 139, including disk and manual.

To order your **SPL** program call us today specifying what disk format you require. You can charge it to your VISA or Master Card if you prefer.

Blat Research+Development Corp.
8016 188th SW, Edmonds, WA 98020
Phone orders: [206] 771-1408

Circle **no. 7** on reader service card.

MCDISPLAY™

$175.00

THE BEST MBASIC DISPLAY INTERFACE EVER DEVELOPED!

Let MCDISPLAY handle the interface to the program user in your application program. For CP/M.

ORDER YOUR COPY TODAY

CALL COLLECT (803) 244-8174

DEMO PACKAGE $10.00 MANUAL $25.00
CHECK, MONEY ORDER, P.O., VISA, MASTERCARD

mj MasterComputing Inc.
P.O. Box 17442
Greenville, SC 29606
(803) 244-8174

CP/M is a trademark of DIGITAL RESEARCH
MBASIC is a product of Microsoft

Circle **no. 51** on reader service card.

for details), but two new commands have been created for the edit mode: the *HOME* key moves alternately to the top and bottom line of the screen, and the *ERASE* key erases from the current cursor position to the end of the line. More subtle changes have also been made. The *RETURN* key moves to the beginning of the next line in edit mode, but acts as the *insert down* key in the insert mode. Both the *DC* and *DELETE* keys delete the character at the cursor, but *BACKSPACE* deletes the character to the left of the cursor, the same as the original *delete character* special key. The only commands remaining as control codes are *split* and *join*. There is some room for expansion: three keys are currently assigned to force command mode, but could be reassigned. Note that the *ESC* key must be pressed twice to be recognized, because it is also issued by the special character codes of the keyboard. My choice of key assignments and features has admittedly been influenced by the PIE editor.

Implementation Details

I'd like to start out with an unsolicited endorsement of the Software Toolworks C/80 Compiler, now in version 2.0 and incorporating almost all of the C language. It's a descendent of Small-C and is inexpensive, fast, and powerful. Its salient feature in the context of the editor is support for modular compilation using Microsoft M80 and L80. Since the editor is broken into nine fairly independent modules, changes can be made in one without having to recompile everything. Also, SID can be used for debugging using global variable and function names. The disadvantage is that variable and function names must be distinct at six characters' length. This has necessitated renaming several functions in the editor, as listed in Figure 1 (page 42). (I have not listed each occurrence of the altered names in the listing; I leave that as a test of the capabilities of your present editor.) Also, to force the main buffer to be at the end of the program, the short program MBUFFER.MAC must be assembled by M80 and be the *last* module linked (after the C/80 CLIBRARY module).

Most details of the changes are explained in the listing on page 43. The functions that are completely unchanged are not included, but a note indicates their position. In general, I have listed the entire function if there has been any change.

Concluding Remarks

This editor is a very useful addition to my stable of inexpensive editors, and is particularly suited to editing large files, interleaving text from other files, and breaking up a file or portions of a file into smaller pieces. It doesn't do everything, and reading and writing to files are considerably slower than for editors written in assembly language. I suggest those with H89's go through the bothersome task of reassembling their BIOS with the type-ahead buffer option so that rapid typing will not lose characters.

When I move on to my next system, I will feel secure that I have its first editor waiting in the wings, and it won't cost me twice as much as my previous editor for the same features. The modifications to the editor are in the public domain, and a somewhat earlier version has been submitted to the SIG/M User Group (Amateur Computer Group of New Jersey, Box 97, Iselin, NJ 08830).

DDJ

(Figure 1 on page 42)

(Listing begins on page 43)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 233**

Figure 1. (Text begins on page 38)

Altered Function Names

Original	New	Original	New
search1	suurch	pmtcoll	pmt1col
outgetx	outxget	sysmovdn	sysdnmov
outgety	outyget	sysmovup	sysupmov
outhasdn	outdnhas	bufdeln	bufndel
outhasup	outuphas	bufoutln	buf1nout
pmtmode1	pmt1mode	bufmovup	bufupmov
pmtfile1	pmt1file	bufmovdn	bufdnmov
pmtline1	pmt1line	buffer	mbuffer

DOCUMENTATION FOR "C" EDITOR – FEBRUARY 28, 1983

CP/M-H19-H89 Version; written by Edward K. Ream; *Dr. Dobb's Journal*, January 1982, V. 7, Issue 1; modified by Alan D. Howard.

COMMAND MODE COMMANDS: (May be entered in upper or lower case)

Command	Arguments	Function
append	<filename>	Insert the named file into buffer at cursor
change	<from to>	Make indicated changes in lines in range <from to>
clear		Erase the buffer
closeread		Close the read file
closewrite		Close the write file
copy	<from to n>	Copy <n> lines from <from> to before <to>
delete	<from to>	Delete all lines in the range <from to>
delname	<filename>	Make the indicated file the write file; no error if file already exists
dos		Exit from editor to operating system
extract	<from to>	Write the designated lines to write file
find		Search for a pattern; enter edit mode
g	<n>	Go to line <n> and enter edit mode
help		List the command mode commands to the screen
hidetab		Do not show tabs in reverse video (default)
list	<from to >	List the indicated lines to printer
load	<filename>	Open the indicated file as the read file; clear the buffer; load the buffer from the file; close the read file if entire file read
move	<from to n>	Move <n> lines from <from> to before <to>
name	<filename>	Make the indicated file the write file; error if file already exists
open	<filename>	Open the indicated file as the read file
read	<n>	Read <n> lines from read file to end of buffer; close the read file if entire file read
rename	<filename>	Close the write file; open new write file
rest		Clear the buffer if requested; load the buffer from read file; close the read file if entire file read

(Continued on page 42, column 2)

Command	Arguments	Function
save		Save the buffer into the write file; buffer is unchanged
search	<from to>	Print all lines that match a pattern
showtab		Show tabs in reverse video
tabs	<n>	Set tab stops at every <n> columns
write	<n>	Write <n> lines from front of buffer to the write file; the lines are deleted from the buffer
(blue key)		Enter edit mode
(IC key)		Enter insert mode

EDIT AND INSERT MODE KEY COMMANDS:
(Commands marked † not valid in insert mode; Commands marked ⸸ force edit mode)

Key	Action
(f1)	† Scroll down until any key pressed
(f2)	† Move to end of line
(f3) <n> CR	† Go to line <n>
(f4)	† Move to beginning of line
(f5)	† Scroll up until any key pressed
(ERASE)	† Erase from cursor to end of line
(Blue)	Enter edit mode
(Red)	Abort changes to current line
(White)	Enter command mode
(BACKSPACE)	Delete character before cursor
(DELETE)	Delete character at cursor; same as (BACKSPACE) at end of line
(RETURN)	Insert mode: add line below current line, move to the new line Edit mode: move to next line
(LINE FEED)	Insert line at cursor: move current line down, enter insert mode
(IC)	Enter insert mode
(DC)	Delete character at cursor; same as (BACKSPACE) at end of line
(IL)	Insert line at cursor: move current line down, enter insert mode
(DL)	Delete current line
(0) <char>	† (Zero on auxiliary keypad); search for character and move cursor to character
(.) <char>	† (Period on auxiliary keypad); delete characters from cursor up to but not including <char>
(ENTER)	Enter command mode
(HOME)	† Move cursor alternately to top and bottom of screen
(Control-S)	Split line at cursor
(Control-P)	Join current line with line above (if room)
(ESC ESC)	Enter command mode
(Up-arrow)	⸸ Up one line
(Down-arrow)	⸸ Down one line
(Left-arrow)	Left one character
(Right-arrow)	Right one character
(All others)	Any other printing key (and TAB key) act as follows: Edit mode: replace cursor with character; Insert mode: insert character

End Figure 1.

(Continued on page 43, column 1)

Screen Editor Enhancements

(Text begins on page 38)

```
/* ED0.C */

/* INCLUSION FILE */

#define MAXLEN   80       /* 80 character video screen */
#define MAXLEN1  81       /* MAXLEN+1 */
#define SYSFNMAX 15
#define EOS      0
#define OK       1
#define ERR      -1
#define EOF      -2
#define YES      1
#define NO       0
#define CR       13
#define LF       10
#define TAB      9
#define HUGE     32000
```

```
/* ED1.CCC */

/* INCLUSION FILE */

#define UP1      10
#define DOWN1    13
#define UP2      21
#define DOWN2    4
#define LEFT1    25       /* new code assignment */
#define RIGHT1   18
#define INS1     14
#define EDIT1    5
#define ESC1     27
#define DEL1     127
#define ZAP1     26
#define ABT1     24
#define SPLT1    19
#define JOIN1    16
#define DTOCH    1        /* kill mode in edit */
#define GTOCH    2        /* search mode in edit */
#define HOME     6        /* home cursor or to bottom of screen */
#define DSCROL   7        /* scroll down */
#define GOTO     17       /* go to line */
#define LSTRT    11       /* go to beginning of line */
#define LEND     12       /* go to end of line */
#define USCROL   15       /* scroll up */
#define ERASE    20       /* erase to end of line */
#define SCRNW    81       /* H19 screen width */
#define SCRNW1   80       /* SCRNW-1 */
#define SCRNL    24       /* H19 screen length */
#define SCRNL1   23       /* SCRNL-1 */
#define SCRNL2   22       /* SCRNL-2 */
#define SCRNL3   13       /* half a screen length */
#define LISTW    80       /* width of listing */
#define LFTDEL    8       /* delete to left of cursor */
```

```
/* ED2.C */

#include ed0.c
#include ed1.ccc
#define SIGNON "E.K. Ream/Dr Dobb's Editor - H89/H19 enhancement: Feb. 20, 1983"
#define HELP1 "open <f/n> : name <f/n> : rename <f/n> : delname <f/n>"
#define HELP2 "load <f/n> : append <f/n> : rest : read <n> : write <n> : save"
#define HELP3 "extract <f t> : closeread : closewrite"
```

(Continued on page 43, column 2)

```
#define HELP4 "clear : delete <f t> : move <f t n> : copy <f t n>"
#define HELP5 "find : search <f t> : change <f t>"
#define HELP6 "tabs <n> : showtab : hidetab : help : dos : g <n>"
#define CMNDMODE        1
#define INSMODE         2
#define EDITMODE        3
#define EXITMODE        4
main(argc,argv) char *argv[];
{
int mode;
int i;
        syscout(ESC1); /*alternate keyboard mode*/
        syscout('=');
        fmtassn(NO);
        fmtset(8);
        fmtntab();
        outclr();
        outxy(0,SCRNL1);
        message(SIGNON);
        outxy(0,1);
        fileclear();    /* make sure no read or write files indicated */
        bufnew();
        mode=CMNDMODE;
        if (argc>1) {
                open(argv[1],NO);
                rest(argv[0]);
                outxy(0,1);
        }
        edgetln();
        while(1){
                if (mode ==EXITMODE) {
                        break;
                }
                else if (mode==CMNDMODE) {
                        mode=command();
                }
                else if (mode==EDITMODE) {
                        mode=edit();
                }
                else if (mode==INSMODE) {
                        mode=insert();
                }
                else {
                        syserr("main: no mode");
                        mode=EDITMODE;
                }
        }
}
edit()
{
char sbuffer[SCRNW1];
int v;
int x,y;
char c;
        pmtedit();
        while(1){
                c=syscin();     /* tolower eliminated */
                if (c==ESC1) {          /* enter command mode */
                        return(CMNDMODE);
                }
                else if (c==INS1) {     /* enter insert mode */
                        return(INSMODE);
                }
                else if (special(c)==YES) {
                        if (c==UP1){    /* DOWN1 now treated separately */
                                return(INSMODE);
                        }
                        else {
                                continue;
                        }
                }
                else if (c==DOWN1) {    /* DOWN1 now does not    */
```

(Continued on page 44, column 1)

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
		eddn();			/* enter insert mode but */
		pmtline();		/* just advances to next line */
		edbegin();
		pmtcol();
	}
	else if (c==RIGHT1) {	/* go to end of line */
		edright();
		pmtcol();
	}
	else if (c==ERASE) {	/* erase to end of line */
		ederase();
		pmtcol();
	}
	else if (c==HOME) {	/* move cursor alternatingly to */
		edhome();		/* top and bottom of screen */
		pmtline();
	}
	else if (c==LSTRT) {	/* move to beginning of line */
		edbegin();
		pmtcol();
	}
	else if (c==DSCROL) {	/* scroll down */
		pmtmode("edit: scroll");
		while (bufnrbot()==NO) {
			if (chkkey()==YES) {
				break;
			}
			if (eddn()==ERR) {
				break;
			}
		}
		pmtedit();
	}
	else if (c==LEND){	/* move to end of line */
		edend();
		pmtcol();
	}
	else if (c==GOTO){	/* go to line entered */
		x=outxget();
		y=outyget();
		pmtcmnd("edit: goto: ",sbuffer);
		if(number(sbuffer,&v)) {
			edgo(v,0);
		}
		else {
			outxy(x,y);
		}
		pmtedit();
	}
	else if (c==DTOCH){	/* kill to character entered */
		pmtmode("edit: kill");
		c=syscin();
		if ((special(c)==NO) &
		    (control(c)==NO)) {
			edkill(c);
		}
		pmtedit();
	}
	else if (c==GTOCH){	/* search to character entered */
		pmtmode("edit: search");
		c=syscin();
		if ((special(c)==NO) &
		    (control(c)==NO)) {
			edsrch(c);
		}
		pmtedit();
			edsplit();
			pmtline();
			return(YES);
		}
		if (c==ABT1) {
			edabt();
			pmtcol();
			return(YES);
		}
		else if (c==LFTDEL) {		/* delete to left of cursor */
			edldel();
			pmtcol();
			return(YES);
		}
		else if (c==DEL1) {		/* delete at cursor */
			edcdel();
			pmtcol();
			return(YES);
		}
		else if (c==ZAP1) {
			edzap();
			pmtline();
			return(YES);
		}
		else if (c==UP2) {
			edup();
			pmtline();
			return(YES);
		}
		else if (c==UP1) {
			ednewup();
			pmtline();
			return(YES);
		}

/* DOWN1 (CR) is no longer a special character */

		else if (c==DOWN2) {
			eddn();
			pmtline();
			return(YES);
		}
		else if (c==LEFT1) {
			edleft();
			pmtcol();
			return(YES);
		}
		else if (c==RIGHT1) {
			edright();
			pmtcol();
			return(YES);
		}
		else	{
			return(NO);
		}
}
command()
{
int v;
char c;
char args[SCRNW1];
char *argp;
int topline;
int ypos;
int oldline;
int k;
	edrepl();
	oldline=bufln();
	ypos=outyget();
	topline=oldline-ypos+1;
	while(1) {
		outxy(0,SCRNL1);
```

```
		}
		else if (c==USCROL) {	/* scroll up */
			pmtmode("edit: scroll");
			while (bufattop()==NO) {
				if (chkkey()==YES) {
					break;
				}
				if (edup()==ERR) {
					break;
				}
			}
			pmtedit();
		}
		else {		/* editor now exchanges any other character */
			if ((special(c)==NO) &	/* with cursor */
			    (control(c)==NO)) {
				edchng(c);
			}
			pmtcol();	/* only need to update column number */
		}
	}
}
insert()
{
char c;
	pmtmode("insert");
	while(1) {
		c=syscin();
		if (c==ESC1) {
			return(CMNDMODE);
		}
		else if (c==EDIT1) {
			return(EDITMODE);
		}
		else if (c==INS1) {
			;
		}
		else if (c==DOWN1) {	/* in insert mode DOWN1 treated	*/
			ednewdn();	/* differently from edit mode	*/
			pmtline();	/* inserts new line */
			continue;
		}
		else if (special(c)==YES) {
			if ((c==UP2)|(c==DOWN2)) {
				return(EDITMODE);
			}
			else {
				continue;
			}
		}
		else if (control(c)==YES) {
			continue;
		}
		else {
			edins(c);
			pmtcol();
		}
	}
}

/***** control(c) unchanged *****/

special(c) char c;
{
int k;
	if (c==JOIN1) {
		edjoin();
		pmtline();
		return(YES);
	}
	if (c==SPLT1) {
```

(Continued on page 44, column 2)

```
			fmtcrlf();
			pmtmode("command:");
			getcmnd(args,0);
			fmtcrlf();
			pmtline();
			c=args[0];
			if ((c==EDIT1)|(c==INS1)) {
				if (oldline==bufln()) {
					edgetln();
					bufout(topline,1,SCRNL1);
					outxy(0,ypos);
				}
				else {
					edgo(bufln(),0);
				}
		if (c==EDIT1) {
			return(EDITMODE);
		}
		else {
			return(INSMODE);
		}
}
else if (tolower(args[0])=='g'){
		argp=skipbl(args+1);
		if (argp[0]==EOS) {
			edgo(oldline,0);
			return(EDITMODE);
		}
		else if (number(argp,&v)==YES) {
			edgo(v,0);
			return(EDITMODE);
		}
		else {
			message("bad line number");
		}
}
else if (lookup(args,"append")) {
		append(args);
}
else if (lookup(args,"change")) {
		change(args);
}
else if (lookup(args,"clear")) {
		clear();
}
else if (lookup(args,"delete")) {
		delete(args);
}
else if (lookup(args,"dos")) {
		if (chkbuf()==YES) {
			closewrite();	/* write file closed on exit */
			syscout(ESC1); /*exit alternate keyboard mode*/
			syscout('>');
			syscout(ESC1); /*clear display*/
			syscout('E');
			return(EXITMODE);
		}
}
else if (lookup(args,"find")) {
		if ((k=find()) >= 0) {
			edgo(bufln(),k);
			return(EDITMODE);
		}
		else {
			bufgo(oldline);
			edgetln();
			message("pattern not found");
		}
}
else if (lookup(args,"list")) {
		list(args);
```

(Continued on page 46, column 1)

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
        }
        else if (lookup(args,"open")) {          /* new */
                open(args,YES);
        }
        else if (lookup(args,"load")) {          /* changed */
                open(args,YES);
                rest(args);
        }
        else if (lookup(args,"name")) {
                name(args);
        }
        else if (lookup(args,"write")) {         /* new */
                writel(args);
        }
        else if (lookup(args,"rename")) {        /* new */
                rename(args);
        }
        else if (lookup(args,"delname")) {       /* new */
                delname(args);
        }
        else if (lookup(args,"closewrite")) {    /* new */
                closewrite(args);
        }
        else if (lookup(args,"showtab")) {     /* new */
                fmtstab();
        }
        else if (lookup(args,"hidetab")) {     /* new */
                fmtntab();
        }
        else if (lookup(args,"closeread")) {     /* new */
                closeread(args);
        }
        else if (lookup(args,"help")) {          /* new */
                outclr();
                outxy(0,SCRNL1);
                message(HELP1);
                message(HELP2);
                message(HELP3);
                message(HELP4);
                message(HELP5);
                message(HELP6);
        }
        else if (lookup(args,"save")) {
                save(args);
        }
        else if (lookup(args,"read")) {           /* new */
                setit(args);
        }
        else if (lookup(args,"move")) {          /* new */
                moveit(args);
        }
        else if (lookup(args,"copy")) {          /* new */
                copyit(args);
        }
        else if (lookup(args,"search")) {
                search(args);
        }
        else if (lookup(args,"tabs")) {
                tabs(args);
        }
        else if (lookup(args,"rest")) {          /* new */
                rest(args);
        }
        else if (lookup(args,"extract")) {       /* new */
                extract(args);
        }
```

```
        }
        while ((bufatbot()==NO)&(bufln()<=to)) {         /* do the writing */
                n=bufgetln(databuf,MAXLEN);
                n=min(n,MAXLEN);
                if (pushline(writefile,databuf,n)==ERR){
                        break;
                }
                if (bufdn()==ERR) {
                        break;
                }
        }
        bufgo(oldline);
}
open (args,flag) char *args; int flag;
{
/* open a file for reading */
char locfn [SYSFNMAX];
int n;
int file;
int topline;
        if (readfile > 0) {                /* check for open readfile */
                message("read file still open");
                return;
        }
        if (flag==YES) {
                if (name1(args,locfn)==ERR) {
                        return;
                }
        }
        else  {
                if (name3(args,locfn)==ERR) {
                        return;
                }
        }
        if (locfn[0]==EOS) {
                message("no file argument");
                return;
        }
        if (chkbuf()==NO) {
                return;
        }
        if ((file=sysopen(locfn,"r"))==ERR) {
                message("file not found");
                return;
        }
        syscopfn(locfn, rfilename);      /* make the file name the readfile */
        readfile = file;
        pmtrfile(rfilename);
        bufnew();
}
setit (args) char *args;          /* new: add n lines to buffer if room */
{
int n;
int topline;
int nlines,npoint;
        if (readfile == -1) {
                message ("no read file");
                return;
        }
        pmtrfile(rfilename);
        if (getarg(args,&nlines)==ERR) {
                return;
        }
        if (nlines<1) {
                return;
        }
        if (bufgo(HUGE)==ERR) {
                return;
        }
        npoint=1;
        while (npoint<=nlines) {         /* add the lines */
                npoint++;
                if ((n=readline(readfile,databuf,MAXLEN))>=0) {
```

```
		else {
			message("command not found");
		}
	}
}

/***** lookup() and setcmnd() unchanged *****/

-----------------------------------------------------------------------------------

/* ED3.C */

#include ed0.c
#include ed1.ccc
int readfile = -1;	/* separate read and write files */
int writefile = -1;	/* -1 for inactive, file channel no. for active */
char rfilename[SYSFNMAX];	/* file names */
char wfilename[SYSFNMAX];
char databuf[MAXLEN1];	/* single buffer replaces redundant buffers */
fileclear()		/* initializes filenames to zero length */
{
	rfilename[0] = EOS;
	wfilename[0] = EOS;
}

/***** append() unchanged *****/

/***** the only alterations to change() are in amatch() references
	1st reference was: if(amatch(oldline,oldpat+1,0)==YES) {
		is now: if (amatch(oldline,oldpat+1,NO)==0) {
	2nd reference was: if (amatch(oldline,oldpat,col++)==YES) {
		is now: if ((col=amatch(oldline,oldpat,YES))>=0) {
*****/

clear()
{
	if (chkbuf()==YES) {
		outclr();
		outxy(0,SCRNL1);
		bufnew();
		message("buffer cleared");
		return(YES);
	}
	else{
		return(NO);
	}
}

/***** delete() unchanged *****/

find()
{
	return(suurch(bufln()+1,HUGE,YES));
}

/***** list() unchanged *****/

extract(args) char *args;	/* write indicated line range to writefile */
{
int oldline, from, to, n;
	if (writefile == -1) {
		message("file not opened");
		return;
	}
	oldline = bufln();
	if (get2args(args,&from,&to)==ERR) {	/* from and to are line range */
		return;
	}
	if (bufgo(from)==ERR){
		return;
```

(Continued on page 46, column 2)

```
			if (n>MAXLEN) {
				message("line truncated");
				n=MAXLEN;
			}
			if (bufins(databuf,n)==ERR) {
				bufgo(1);
				topline=max(1,bufln()-SCRNL2);
				bufout(topline,2,SCRNL2);
				bufgo(topline);
				return;
			}
			if (bufdn()==ERR) {
				break;
			}
		}
		else {
			npoint = HUGE;	/* reached end of file - close it */
			sysclose(readfile);
			readfile = -1;
			rfilename[0] = EOS;
			pmtrfile(rfilename);
		}
	}
	bufgo(1);
	topline=max(1,bufln()-SCRNL2);
	bufout(topline,2,SCRNL2);
	bufgo(topline);
}
rest (args) char *args; /* new: try to read rest of readfile into buffer */
{
int n;
int topline;
	if (readfile == -1) {
		message ("no read file");
		return;
	}
	pmtrfile(rfilename);
/* if buffer has been changed and not saved, give option of clearing, */
/* otherwise add to end of buffer */
	if(bufchng()==YES) {
		fmtsout("buffer not saved, clear? ",0);
		pmtline();
		if (tolower(syscout(syscin()))=='y'){
			outclr();
			outxy(0,SCRNL1);
			bufnew();
			message("buffer cleared");
		}
		else {
			bufgo(HUGE);
		}
	}
	else{
		outclr();
		outxy(0,SCRNL1);
		bufnew();
		message("buffer cleared");
	}
	while ((n=readline(readfile,databuf,MAXLEN))>=0) {
		if (n>MAXLEN) {
			message("line truncated");
			n=MAXLEN;
		}
		if (bufins(databuf,n)==ERR) {
			bufgo(1);
			topline=max(1,bufln()-SCRNL2);
			bufout(topline,2,SCRNL2);
			bufgo(topline);
			return;
		}
		if (bufdn()==ERR) {
			break;
```

(Continued on page 50, column 1)

Now for Concurrent CP/M-86

VEDIT - The Clear Choice for Programmers

Plus Features for Fast & Efficient Word Processing

Increasing your productivity is what a good text editor is all about. VEDIT excels by giving you a unique combination of extensive and easy to use editing features, customizability and complete hardware support. So compare VEDIT. You'll find everything you expect in a good editor plus a variety of time saving features which only VEDIT offers.

VEDIT is fully user oriented. You can use the function keys on any keyboard, or a layout you are already familiar with - simplifying your usage and easing your learning. While most editors lose text if you run out of disk space, VEDIT lets you delete files or change disks. VEDIT is the result of continuous enhancement and feedback from our nearly 4000 users.

VEDIT
CP/M
CUSTOMIZABLE
FULL SCREEN EDITOR

For program development it surpasses any other editor - with more extensive file handling, important command macro capability and special features for Pascal, PL/1, 'C', Cobol, Assembler and others. With VEDIT you will reduce your program editing time by 30% as compared to the best word processor.

For word processing, VEDIT has word wrap, adjustable margins, reformatting of paragraphs, word and paragraph functions and simple printing with imbedded printer control characters.

Command macros let you perform editing tasks you might otherwise not even attempt. Time consuming tasks for other editors (such as translations or extensive search/replace on many files), can be done by VEDIT without your intervention, even overnight if you choose.

VEDIT supports all of the new CRT terminals, video boards and 8080, Z80 and 8086 computers. We have been consistently first to support new computers - first for CP/M-86, first for MSDOS. And we will support you with any technical assistance you may need.

For the full story, purchase VEDIT risk free. Evaluate the 125 page manual and if you are not satisfied, return the package (disk unopened) for a courteous refund.

CP/M and MP/M are registered trademarks of Digital Research Inc. WordStar and WordMaster are registered trademarks of MicroPro International Corporation. Apple II is a registered trademark of Apple Computer, Inc. MS-DOS and Softcard are trademarks of Microsoft. TRS-80 is a trademark of Tandy Corporation. IBM is a trademark of International Business Machines.

COMPARE VEDIT

Feature	VEDIT	WordMaster	WordStar
True Full Screen Editing	Yes	Yes	Yes
Edit files one disk in length	Yes	Yes	Yes
Compact and fast	Yes	Yes	No
Display of line and column #	Yes	No	Yes
Set/Goto text markers	Yes	No	Yes
'Undo' key to restore line	Yes	No	No
Automatic Indent/Undent	Yes	No	No
Adjustable tab positions	Yes	No	Yes
Repeat function key	Yes	Yes	No
Text move and copy	Yes	Yes	Yes
Scratchpad buffers	10	Only 1	No
Load/Save buffers on disk	Yes	No	No
Flexible command mode	Yes	Yes	No
Multiple command macros	Yes	No	No
Directory display	Yes	No	Yes
Edit additional (small) files simultaneously	Yes	No	No
Insert another disk file	Yes	Yes	Yes
Unlimited file handling	Yes	No	No
Automatic disk buffering	Yes	Yes	Yes
Recovery from 'Full Disk'	Yes	No	Some
Change disks while editing	Yes	No	No
Startup command file	Yes	No	No
Program CRT function keys	Yes	No	No
Word Wrap and reformatting	Yes	No	Yes
Printing	Simple	No	Extensive
Print formatting	No	No	Yes
Menu driven installation	Yes	No	Yes
Support newest CRT terminals	Yes	No	No
Support smart CRT functions	Yes	No	Some
Customizable keyboard layout	Yes	No	No
Available for CP/M-86	Since 1981	?	?
Available for MSDOS	Since 1981	?	Yes

Please specify your microcomputer, video board or the CRT terminal version, 8080, Z80, or 8086 code, operating system and disk format.

VISA & MasterCard

VEDIT - Disk and Manual
For 8080, Z80 or IBM PC $150
For CP/M-86 or MSDOS $195
Manual only $18

Zenith Z100 and Z89 • DEC VT180 • Televideo 802
TRS-80 I, II and 16 • Xerox 820 • Apple II Softcard
SuperBrain • NorthStar • Cromemco • Altos • Vector
MP/M • CP/M-86 • MP/M-86 • MSDOS • PCDOS

IBM Personal Computer and IBM Displaywriter

CompuView
PRODUCTS, INC.

1955 Pauline Blvd., Suite 200 • Ann Arbor, Michigan 48103 • (313) 996-1299

IN AUSTRALIA DISTRIBUTED BY SOFTWARE SOURCE PTY. LTD.
89 OXFORD ST., BONDI JUNCTION, SYDNEY - (02) 389-6388

Circle **no. 19** on reader service card.

Screen Editor Enhancements

(Listing continued, text begins on page 38)

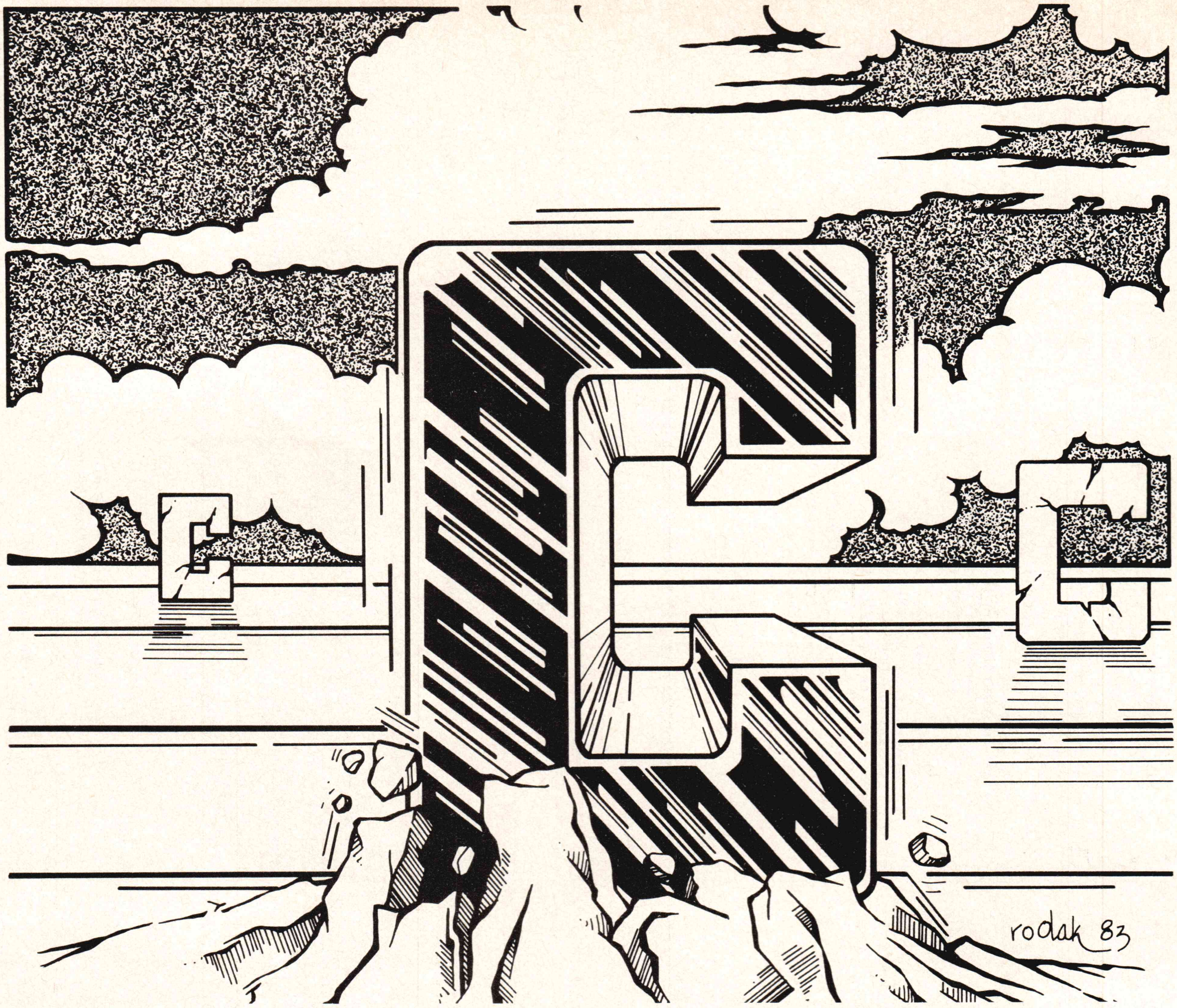
```
                        }
                }
                sysclose(readfile);     /* close file if all read */
                readfile = -1;
                rfilename[0] = EOS;
                pmtrfile(rfilename);
                bufgo(1);
                topline=max(1,bufln()-SCRNL2);
                bufout(topline,2,SCRNL2);
                bufgo(topline);
}
rename(args) char *args;        /* new: change name of writefile */
{
        if (writefile == -1) {
                message("no write file - call name");
                return;
        }
        sysclose(writefile);
        writefile = -1;
        wfilename[0]=EOS;
        pmtwfile(wfilename);
        if (name0(args,wfilename)==ERR) {
                return;
        }
        pmtwfile(wfilename);
}
name(args) char *args;          /* altered definition: name the writefile */
{
        if (writefile > 0) {
                message("write file open");
                return;
        }
        if (name0(args,wfilename)==ERR){
                return;
        }
        pmtwfile(wfilename);
}
name0(args,wfilename) char *args, *wfilename;   /* names new writefile */
{
int file;
        args=skiparg(args);
        args=skipbl(args);
        if (syschkfn(args)==ERR) {
                return(ERR);
        }
        syscopfn(args,wfilename);
        if ((file=sysopen(wfilename,"r"))!=ERR) {
                sysclose(file);
                message("disk file exists");
                return(ERR);
        }
        if ((file=sysopen(wfilename,"w"))==ERR) {
                return(ERR);
        }
        writefile = file;
        return(OK);
}
delname(args) char *args;       /* new: similar to old resave command, */
{                               /* but doesn't automatically write file */
        if (writefile > 0) {
                message("write file open");
                return;
        }
        if (name2(args,wfilename)==ERR){
                return;
        }
        pmtwfile(wfilename);
        }
        oldline=bufln();
        if (bufgo(1)==ERR) {
                sysclose(writefile);    /* close the writefile if error */
                writefile = -1;
                wfilename[0]=EOS;
                pmtwfile(wfilename);
                return;
        }
        while (bufatbot()==NO) {
                n=bufgetln(databuf,MAXLEN);
                n=min(n,MAXLEN);
                if (pushline(writefile,databuf,n)==ERR) {
                        break;
                }
                if (bufdn()==ERR) {
                        break;
                }
        }
        if (bufatbot()) {
                bufsaved();
        }
        bufgo(oldline);
                                        /* writefile no longer closed */
}
closeread()                             /* new: close the readfile */
{
        if (readfile!= -1) {
                sysclose(readfile);
                readfile= -1;
                rfilename[0]=EOS;
                pmtrfile(rfilename);
        }
}
closewrite()                            /* new: close the writefile */
{
        if (writefile != -1) {
                sysclose(writefile);
                writefile= -1;
                wfilename[0]=EOS;
                pmtwfile(wfilename);
        }
}

/***** search() unchanged *****/

/***** the only alterations to search1() are in amatch() references
        1st reference was: if(amatch(line,pat+1,0)==YES) {
                   is now: if (amatch(line,pat+1,NO)==0) {
        2nd reference was: if (amatch(line,pat,col++)==YES) {
                   is now: if ((col=amatch(line,pat,YES))>=0) {
        Also - search1() is now called suurch()
*****/

/***** tabs() and chkbuf() are unchanged *****/

getarg(args,val) char *args; int *val;  /* new: get a single argument */
{
        args=skiparg(args);
        args=skipbl(args);
        if (*args==EOS) {
                *val=1;
                return(ERR);
        }
        if (number(args,val)==NO) {
                message("bad argument");
                return(ERR);
        }
        return(OK);
}
/* new: get three arguments                                              */
get3args(args,val1,val2,val3) char *args; int *val1, *val2,*val3;
```

```
}
name2(args,wfilename) char *args, *wfilename;    /* open writefile: no error */
{                                                /* file exists */
int file;
        args=skipargs(args);
        args=skipbl(args);
        if (syschkfn(args)==ERR) {
                return(ERR);
        }
        syscopfn(args,wfilename);
        if ((file=sysopen(wfilename,"w"))==ERR) {
                return(ERR);
        }
        writefile = file;
        return(OK);
}

/***** name1() unchanged *****/

name3(args,filename) char *args, *filename;
{
        if (syschkfn(args)==ERR) {
                return(ERR);
        }
        syscopfn(args,filename);
        return(OK);
}
writel(args) char *args;        /* new: write first n lines of buffer to */
{                               /* writefile and delete from buffer */
int n, to;
        if (writefile == -1) {
                message("file not opened");
                return;
        }
        if (setargs(args,&to)==ERR) {
                return;
        }
        if (bufgo(1)==ERR) {
                sysclose(writefile);
                writefile = -1;
                wfilename[0] = EOS;
                pmtwfile(wfilename);
                return;
        }
        while ((bufatbot()==NO)&(bufln()<=to)) {
                n=bufgetln(databuf,MAXLEN);
                n=min(n,MAXLEN);
                if (pushline(writefile,databuf,n)==ERR) {
                        return;
                }
                if (bufdn()==ERR) {
                        return;
                }
        }
        if (bufgo(1)==ERR) {
                return;
        }
        if (bufndel(to)==ERR){
                return;
        }
        bufout(bufln(),1,SCRNL1);
        if (bufatbot()) {
                bufsaved();
        }
        bufgo(1);
}
save()
{
int n, oldline;
        if (writefile == -1) {                  /* changed to writefile */
                message("file not opened");     /* file assumed already open */
                return;
```

(Continued on page 50, column 2)

```
{
        args=skipargs(args);
        args=skipbl(args);
        if (*args==EOS) {
                return(ERR);
        }
        if (number(args,val1)==NO) {
                message("bad argument");
                return(ERR);
        }
        args=skipargs(args);
        args=skipbl(args);
        if(*args==EOS) {
                return(ERR);
        }
        if (number(args,val2)==NO) {
                message("bad argument");
                return(ERR);
        }
        args=skipargs(args);
        args=skipbl(args);
        if(*args==EOS) {
                return(ERR);
        }
        if (number(args,val3)==NO) {
                message("bad argument");
                return(ERR);
        }
        else {
                return(OK);
        }
}

/***** set2args(), skipargs(args), and skipbl() unchanged *****/

chkkey()
{
int c;
        c=syscstat();
        if (c==-1) {            /* note correction from original version */
                return(NO);
        }
        else if (c==' ') {
                pmtline();
                if (syscin()==' ') {
                        return(NO);
                }
        }
        return(YES);
}

/***** amatch() is now machine language code in ED8.c *****/

/***** replace() is unchanged *****/

copyit (args) char *args;       /* new: copies "lenn" lines from "from"   */
{                               /* to "to" */
int from,to,lenn;
int topline;
        if (set3args(args,&from,&to,&lenn)==ERR) {
                return;
        }
        if (to == from) {
                return;
        }
        if ((to>from)&(to<=(from+lenn))) {
                message("interleaving not permitted");
                return;
        }
        bufcopy(from,to,lenn);
        bufgo(1);
```

(Continued on page 54, column 1)



"Introducing the new Eco-C Compiler"

You already know C has what you need in a language; structured code that rivals assembler in size and speed, a rich set of operators and the flexibility that you demand. And C's portability means that your software won't become obsolete. No more learning a new instruction set each time a new processor pops up.

Only problem was that a full-featured C compiler either cost a fortune or it lacked the data types you need, like floating point numbers. Meet the answer to your problem, our new Eco-C™ compiler.

Eco-C is a full C compiler with a complete set of operators and data types including longs, floats and doubles. You don't have to settle for less.

We also know your time is valuable, and you've got better things to do than wait for a compile-link to end. So, we designed the compiler from the ground up. It's based on a true LL(1) grammar; no brute force parsing. A typical compile-and-link takes only a minute or two. Error messages are meaningful and right on the money.

And when your masterpiece is done, it's fast, efficient and **yours.** There are no royalty fees on software produced with the Eco-C compiler.

Everything that you need is included. We've teamed Eco-C with Microsoft's MACRO 80™ to give you a reliable assembler and linker; a $200.00 value by itself! Since the compiler generates assembler output, you can even modify the compiled code if you wish. You can then assemble it (M80) to produce REL files for the linker (L80). Of course there's a run-time library, and helpful user's manual, too.

Eco-C is designed for the Z80™ CPU using either CP/M or MP/M. (Other versions coming soon.) The price is $350.00 and the user's manuals are $40.00. For more information, call or write:

ECOSOFT INC.

P.O. Box 68602
Indianapolis, IN 46268
(317) 255-6476

VISA

master charge

IS YOUR NEW CP/M™ MICRO STILL INTIMIDATING YOU?

By now, you may have discovered that CP/M is a wonderful operating system. But not an easy one to work with.

Here at last is **POWER!**, a super-power-packed, user-friendly program that takes the frustration and intimidation out of using CP/M.

POWER! offers 55 prompted, easy-to-use CP/M utilities in one 15k package. It automatically numbers disk files. From a screen menu, you pick the number to Copy, Erase, Reclaim, Rename, Type, etc. Your computer feeds the file names automatically. No more typing errors. And you can even run programs from the numbered menu.

No more BDOS errors. And you don't need a system disk in any disk drive!

POWER! lets you Test and Fix bad disks. Reclaim accidentally erased files or programs. Single step through memory up or down. Search, View, Change memory or disk in a snap. See Status and File Size instantly. Verify Checksums for programs. Load or Save programs at any address. Convert Hex numbers to Decimal or Binary instantly.

Now, **POWER!** also includes a special program that lets you lock sensitive files, so that only you can access them. Without the secret PASSWORD which you can create and change at will, no prying eyes will ever know your secret file even exists.

POWER! will operate in any standard CP/M or MP/M system, including CP/M-86, IBM PC, Apple (Z80 card), Osborne, Kaypro, HP, TeleVideo, TRS-80 conversions, S100's including NorthStar, Vector, Morrow, CompuPro, etc. Available in all disk formats.

InfoWorld
Software Report Card

Power Version 2.55

	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☑
Documentation	☐	☐	☑	☐
Ease of Use	☐	☐	☐	☑
Error Handling	☐	☐	☐	☑

It comes with a 120-page easy-read guide. And at only $3.00 per utility command, **POWER!** is your best buy in software. It's 55 friendly programs rolled into one to simply put you in control of CP/M.

Don't be intimidated by CP/M. Call or send in your order today and **Take The POWER! Trip.**

Only $169. Money Back Guarantee

Charge & COD Orders Welcome VISA MasterCard

TOLL FREE (800)428-7825 Ext. 96K
IN CA: (800)428-7824 Ext. 96K
DEALERS AND OEM'S (415)567-1634 Ext.K

Take The 'POWER!' Trip.

COMPUTING!
2519 Greenwich San Francisco, CA 94123
TOLL FREE (800)428-7825 Ext. 96K
IN CA: (800)428-7824 Ext. 96K
DEALERS AND OEM'S (415)567-1634 Ext.K

ONLY $169. Calif. add 6½% sales tax
☐ **CP/M $169.** ☐ **CP/M-86 $169.** ☐ **MP/M $249.**

Card No. __________
Exp. Date __________
Computer __________
Your Name __________
Company Name __________
Address __________
City/State/Zip __________

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
	topline=max(1,bufln()-SCRNL2);
	bufout(topline,2,SCRNL2);
	bufgo(topline);
}
moveit (args) char *args;	/* new: moves 'lenn' lines from 'from' */
{				/* to 'to' - same as copyit except deletes */
int from,to,lenn;		/* old lines */
int topline;
	if (get3args(args,&from,&to,&lenn)==ERR) {
		return;
	}
	if (to == from) {
		return;
	}
	if ((to>from)&(to<=(from+lenn))) {
		message("interleaving not permitted");
		return;
	}
	bufcopy(from,to,lenn);
	if (to>from){
		bufgo(from);
		bufndel(lenn);
	}
	else {
		bufgo(from+lenn);
		bufndel(lenn);
	}
	bufgo(1);
	topline=max(1,bufln()-SCRNL2);
	bufout(topline,2,SCRNL2);
	bufgo(topline);
}
------------------------------------------------------------------------------

/* ED4.C */

#include ed0.c
#include ed1.ccc
char	editbuf[MAXLEN];
int	editp;
int	editpmax;
int	edcflag;

/***** edabt() and edbegin() unchanged *****/

ederase()	/* new: erase from cursor to end of line */
{
	if (editp==editpmax) {
		return;
	}
	edcflag=YES;
	editpmax=editp;
	edredraw();
}
edhome()	/* new: home cursor.  subsequent calls alternate */
{		/* cursor to top and bottom of screen */
int ypos;
	if (edrepl()!=OK){
		return;
	}
	if ((ypos=outyget())<SCRNL3) {
		while((ypos<SCRNL1)&&(!bufnrbot())){
			if (bufdn()!=OK){
				return(ERR);
			}
			ypos++;
		}
```

```
#include ed0.c
#include ed1.ccc
int fmtrev;
int fmttab;
int fmtdev;
int fmtwidth;
int fmtcol[MAXLEN1];
fmtytab()	/* new - sets mode to show tabs in reverse video */
{
	fmtrev=YES;
}
fmtntab()	/* new - sets normal tab mode (default) */
{
	fmtrev=NO;
}

/***** fmtassn(),fmtadj(), and fmtlen() unchanged *****/
fmtsubs(buf,i,j) char *buf; int i,j;
{
int k;
	if (fmtcol[i]>=fmtwidth) {
		return;
	}
	outxy(fmtcol[i],outyget());
	while (i<j) {
		if (buf[i]==CR) {
			break;
		}
		if (fmtcol[i+1]>fmtwidth) {
			break;
		}
		fmtoutch(buf[i],fmtcol[i]);
		i++;
	}
	if (fmtcol[i]<SCRNW1) {	/* modified to allow full screen width */
	outdeol();
	}
}
fmtsout(buf,offset) char *buf; int offset;
{
char c;
int col,k;
	col=0;
	while (c= *buf++) {
		if (c==CR) {
			break;
		}
		k=fmtchlen(c,col);
		if ((col+k+offset)>fmtwidth) {
			break;
		}
		fmtoutch(c,col);
		col=col+k;
	}
	return(col);		/* fmtsout() now returns column value */
}
fmtchlen(c,col) char c; int col;
{
	if (c==TAB) {
		return(fmttab-(col%fmttab));
	}
	else if ((c<32)&(fmtdev==NO)) {	/* if tabs not displayed */
		return(2);
	}
	else {
		return(1);
	}
}
fmtoutch(c,col) char c; int col;
{
int k;
```

```
	}
	else {
		while((ypos>1)&&(!bufattop())){
			if (bufup()!=OK){
				return(ERR);
			}
			ypos--;
		}
	}
	edsetln();
	outxy(edxpos(),ypos);
}
edchng(c) char c;
{
char oldc;
int k;
	if (editp>=editpmax) {
		edins(c);
		return;
	}
	oldc=editbuf[editp];
	editbuf[editp]=c;
	fmtadj(editbuf,editp,editpmax);
	k=fmtlen(editbuf,editpmax);
	if (k>SCRNW1) {
		editbuf[editp]=oldc;
		fmtadj(editbuf,editp,editpmax);
	}
	else {
		edcflag=YES;
		editp++;
		if (c==TAB) {      /* only need to redraw if TAB char */
		edredraw();
		}
		else {
		fmtchdev(c);
		}
	}
}
/***** eddel() in original version now called edldel(): no change in code *****/

edcdel()  /* deletes character at cursor - edldel() deletes prior character */
{
int k;
	if (edexpos() < outxget()) {
		outxy(outxget()-1,outyget());
		return;
	}
	edcflag=YES;
	if (editp == editpmax)  {
		if (editp==0) {
			return;
		}
		editp--;
		editpmax--;
		edredraw();
		return;
	}
	k=editp;
	while (k<(editpmax-1)) {
		editbuf[k]=editbuf[k+1];
		k++;
	}
	editpmax--;
	edredraw();
}

/***** all remaining ED4.C functions unchanged *****/
--------------------------------------------------------------------------------

/* ED5.C */
```

(Continued on page 54, column 2)

```
	if (c==TAB) {
		k=fmtchlen(TAB,col);
		if ((fmtdev==NO)&(fmtrev==YES)) {
/*show tab characters in reverse video */
			outchar(ESC1);
			outchar('p');
		}
		while ((k--)>0) {
			fmtchdev(' ');
		}
		if ((fmtdev==NO)&(fmtrev==YES)) {
			outchar(ESC1);
			outchar('q');
		}
	}
	else if (c<32) {
/* show control characters in reverse video */
		if (fmtdev==NO) {
			outchar(ESC1);
			outchar('p');
			fmtchdev(c+64);
			outchar(ESC1);
			outchar('q');
		}
		else {
			fmtchdev('^');
			fmtchdev(c+64);
		}
	}
	else {
		fmtchdev(c);
	}
}

/***** fmtchdev(), fmtcrlf(), and fmtset() unchanged *****/
--------------------------------------------------------------------------------

/* ED6.C */

#include ed0.c
#include ed1.ccc
int outx,outy;
/***** outsetx() and outsety() changed in name to outxset() and outyset()***/

/***** outchar() unchanged *****/

outxy(x,y) int x,y;
{
	outx=x;
	outy=y;
	syscout(27);		/* H19/H89 values */
	syscout('Y');
	syscout(y+32);
	syscout(x+32);
}
outclr()
{
	syscout(27);		/* H19/H89 */
	syscout('E');
}
/***** outdelln() unchanged *****/

outdeol()
{
	syscout(27);		/* H19/H89 */
	syscout('K');
}
outuphas()
{
	return(YES);
```

(Continued on page 58, column 1)

GREAT DRIVES. GREAT PRICES.

Floppy Disk Services offers both.

We supply the industry with high quality disk drives and peripherals. And our prices are some of the best you'll see. Floppy Disk Services is a contracted dealer for Siemens, Tandon and Shugart. Starting with their drives, we design complete packages to give you the system capacity you're looking for and the dependability you need.

We offer add-on drives for IBM, Radio Shack, Heath, Apple and most other microcomputers, all at a significant savings to you. Check the examples below and you'll see what we mean.

Apple II Add on drives	$300.00
Apple 8 inch controller	315.00
Apple dual 8 inch system w/controller	1165.00
FDD-100-5b 'flippy' exact HEATH add on	215.00
FDD-200-5 double sided 40 track drive	250.00
SA 455 half hgt DS/DD 48TPI ... $295.00 ..	2 for $245.00 each
SA 465 half hgt DS/DD 96TPI ... $350.00 ..	2 for $295.00 each
FDD-221-5 5ms step 80 track DD/DS	330.00
FDD-100-8d 8" single side DD drive .... $240. ...	2 / $225. each
FDD-200-8p Double sided 8" drive .... $340.	2 / $325. each
TM-100-2 (IBM)	265.00
SA-860 DS/DD Half Hgt 8" .... $495.00 ...	2 for $445.00 each
System packages available for all drives	
Dual 8 inch system with EVERYTHING	750.00*
Dual double sided 8 inch system	950.00*
Single 5¼ Heath or MOD I Add on w/case	265.00*
Dual 5¼ Heath or MOD I	505.00*
10mb Hard Disk for any computer	2300.00*
Magnolia controller, allows any combo 8 and 5¼ inch drives to be added to your H88 or H89	525.00

* 8 inch systems are fully assembled and tested, however the drives are shipped separately from the case to comply with UPS weight restrictions. All 5¼ inch systems come assembled and tested. Prices and specs subject to change without notice.

There's much more. If you don't see what you want, give us a call between 9 am and 5 pm (ET). Chances are we'll have what you need at your price.

Featuring the Shugart Thinline Series.

Floppy Disk Services carries the new Shugart Thinline Dual 8 Inch Drive (Model SA-860). It's a double-density drive, with dual head for a storage capacity of 1.25 megabytes per drive. The system includes our custom cabinets, comes fully assembled and tested, and uses the Power One CP-206 supply, the standard of the industry. Special options available: external drive select using DIP switches, and cable connector on the rear of the cabinet for ease in connecting to your system. Floppy Disk Services designs and builds these systems from the ground up to maximize efficiency and minimize space requirements. Available with Tandon Drives also.

Custom Enclosures. At Floppy Disk Services, we also sell disk drive enclosures, designed by our own experts to be functional and attractive. And our quantity pricing is so competitive, we invite dealers and group purchasers to call.

Thinline/Half Height Specials. Floppy Disk Services has just contracted with Shugart, and we're carrying their half-height 5¼ & 8" systems. If you would like some unbelievable prices on SA-455, SA-465, SA-860's and others, call today! We are legitimate contracted dealers—no middle-man.

Due to production deadlines, prices in this ad are 2 months old, so we encourage you to call us for current prices and new product info.

PAYMENT POLICY — We accept MasterCard, VISA, personal checks & MO. We reserve the right to wait 10 working days for personal checks to clear your bank before we ship. All shipping standard UPS rates plus shipping & handling. NJ residents must add 6% tax.

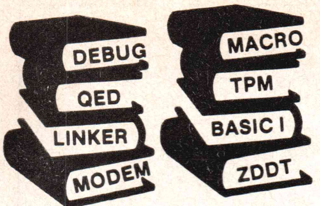
Free Catalogue. If you'd like to receive our Catalogue of Disk Drives and Peripherals, just call or write — we'll mail your copy immediately. And if you want to talk with an expert about getting more out of your system, we'll be happy to help.

Toll Free Order Line **(800) 223-0306**
Tech Help or Info. **(609) 799-4440**

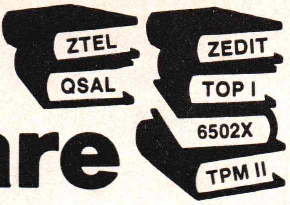
FLOPPY DISK SERVICES™ INC.

741 Alexander Road Princeton, NJ 08540

Circle no. 31 on reader service card.



Z80 Software



SOFTWARE DESCRIPTIONS

TPM (TPM I) - $80 A Z80 only operating system which is capable of running CP/M programs. Includes many features not found in CP/M such as independent disk directory partitioning for up to 255 user partitions, space, time and version commands, date and time, create FCB, chain program, direct disk I/O, abbreviated commands and more! Available for North Star (either single or double density), TRS—80 Model I (offset 4200H) or II, Versafloppy I, or Tarbell I.

TPM-II - $125 An expanded version of TPM which is fully CP/M 2.2 compatible but still retains the extra features our customers have come to depend on. This version is super FAST. Extended density capability allows over 600K per side on an 8" disk. Available preconfigured for Versafloppy II (8" or 5"), Epson QX-10, Osborne II or TRS-80 Model II.

CONFIGURATOR I
This package provides all the necessary programs for customizing TPM for a floppy controller which we do not support. We suggest ordering this on single density (8SD).
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), LINKER ($80), DEBUG I ($80), QED ($150), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$815 Value — **NOW $250**

CONFIGURATOR II
Includes: TPM-II ($125), Sample PIOS (BIOS) SOURCE ($FREE), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QSAL ($200), QED ($150), ZTEL ($80), TOP II ($100), BUSINESS BASIC ($200) and MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1485 Value — **NOW $400**

MODEL I PROGRAMMER
This package is only for the TRS-80 Model I. Note: These are the ONLY CDL programs available for the Model I. It includes: TPM I ($80), BUSINESS BASIC ($200), MACRO I ($80), DEBUG I ($80), ZDDT ($40), ZTEL ($80), TOP I ($80) and MODEM ($40)
$680 Value — **NOW $175**

MODEL II PROGRAMMER
This package is only for the TRS-80 Model II. It includes: TPM-II ($125), BUSINESS BASIC ($200), MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), QED ($150), ZTEL ($80), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM ($40), MODEM SOURCE ($40) and DISASSEMBLER ($80)
$1445 Value — **NOW $375**

BASIC I - $50, a 12K+ basic interpreter with 7 digit precision.

BASIC II - $100, A 12 digit precision version of Basic I.

BUSINESS BASIC - $200, A full disk extended basic with random or sequential disk file handling and 12 digit precision (even for TRIG functions). Also includes PRIVACY command to protect source code, fixed and variable record lengths, simultaneous access to multiple disk files, global editing, and more!

ACCOUNTING PACKAGE - $300, Written in Business Basic. Includes General Ledger, Accounts Receivable/Payable, and Payroll. Set up for Hazeltine 1500 terminal. Minor modifications needed for other terminals. Provided in unprotected source form.

MACRO I - $80, A Z80/8080 assembler which uses CDL/TDL mnemonics. Handles MACROs and generates relocateable code. Includes 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical ops, local and global symbols, linkable module generation, and more!

MACRO II - $100, An improved version of Macro I with expanded linking capabilities and more listing options. Also internal code has been greatly improved for faster more reliable operation.

MACRO III - $150, An enchanced version of Macro II. Internal buffers have been increased to achieve a significant improvement in speed of assembly. Additional features include line numbers, cross reference, compressed PRN files, form feeds, page parity, additional pseudo-ops, internal setting of time and date, and expanded assembly-time data entry.

DEVELOPER I
Includes: MACRO I ($80), DEBUG I ($80), ZEDIT ($50), TOP I ($80), BASIC I ($50) and BASIC II ($100)
$440 Value — **NOW $150**

DEVELOPER II
Includes: MACRO II ($100), MACRO III ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), BUSINESS BASIC ($200), QED ($150), TOP II ($100), ZDDT ($40), ZAPPLE SOURCE ($80), MODEM SOURCE ($40), ZTEL ($80), and DISASSEMBLER ($80).
$1280 Value — **NOW $350**

DEVELOPER III
Includes: QSAL ($200), QED ($150), BUSINESS BASIC ($200), ZTEL ($80) and TOP II ($100)
$730 Value — **NOW $300**

COMBO
Includes: DEVELOPER II ($1280), ACCOUNTING PACKAGE ($300), QSAL ($200) and 6502X ($150)
$1930 Value — **NOW $500**

LINKER - $80, A linking loader for handling the linkable modules created by the above assemblers.

DEBUG I - $80, A tool for debugging Z80 or 8080 code. Disassembles to CDL/TDL mnemonics compatible with above assemblers. Traces code even through ROM. Commands include Calculate, Display, Examine, Fill, Goto, List, Mode, Open File, Put, Set Wait, Trace, and Search.

DEBUG II - $100, A superset of Debug I. Adds Instruction Interpreter, Radix change, Set Trap/Conditional display, Trace options, and Zap FCB.

6502X - $150, A 6502 cross assembler. Runs on the Z80 but assembles 6502 instructions into 6502 object code! Similar features as our Macro assemblers.

QSAL - $200, A SUPER FAST Z80 assembler. Up to 10 times faster than conventional assemblers. Directly generates code into memory in one pass but also to offset for execution in its own memory space. Pascal like structures: repeat...until, if...then...else, while...do, begin...end, case...of. Multiple statements per line, special register handling expressions, long symbol names, auto and modular assembly, and more! This one uses ZILOG Mnemonics.

QED - $150, A screen editor which is both FAST and easy to learn. Commands include block delete, copy, and move to a named file or within text, repeat previous command, change, locate, find at start of line, and numerous cursor and window movement functions. Works with any CRT having clear screen, addressable cursor, clear to end of line, clear to end of screen, and 80X24.

ZTEL - $80, An extensive text editing language and editor modelled after DEC's TECO.

ZEDIT - $50, A mini text editor. Character/line oriented. Works well with hardcopy terminals and is easy to use. Includes macro command capability.

TOP I - $80, A Text Output Processor for formatting manuals, documents, etc. Interprets commands which are entered into the text by an editor. Commands include justify, page number, heading, subheading, centering, and more.

TOP II - $100, A superset of TOP I. Adds: embedded control characters in the file, page at a time printing, selected portion printing, include/merge files, form feed/CRLF option for paging, instant start up, and final page ejection.

ZDDT - $40, This is the disk version of our famous Zapple monitor. It will also load hex and relocatable files.

ZAPPLE SOURCE - $80, This is the source to the SMB ROM version of our famous Zapple monitor. It can be used to create your own custom version or as an example of the features of our assemblers. Must be assembled using one of our assemblers.

MODEM - A communication program for file transfer between systems or using a system as a terminal. Based on the user group version but modified to work with our SMB board or TRS-80 Models I or II. You must specify which version you want.

MODEM SOURCE - $40, For making your own custom version. Requires one of our Macro Assemblers.

DISASSEMBLER - $80, Does bulk disassembly of object files creating source files which can be assembled by one of our assemblers.

HARDWARE

S-100 — SMB II Bare Board $50, "System Monitor Board" for S-100 systems, 2 serial ports, 2 parallel ports, cassette interface, 4K memory (ROM, 2708 EPROM, 2114 RAM), and power on jump. When used with Zapple ROM below, it makes putting a S-100 system together a snap.
Zapple ROM $35, Properly initializes SMB I/II hardware, provides a powerful debug monitor.

IBM PC — Big Blue Z80 board $595, Add Z80 capability to your IBM Personal Computer. Runs CP/M programs but does not require CP/M or TPM. Complete with Z80 CPU, 64K add on memory, serial port, parallel port, time and date clock with battery backup, hard disk interface, and software to attach to PC DOS and transfer programs. Mfr'd by QCS.
50% Discount on all CDL software ordered at the same time as a Big Blue (and for the Big Blue).

APPLE II — Chairman Z80 $345, Add Z80 capability to your Apple II/II Plus computer. Runs CP/M programs with our more powerful TPM. Includes 64K memory add on (unlike the competition this is also useable by the 6502/DOS as well as the Z80), TPM, QSAL assembler, QED Screen Editor, and Business Basic. Mfr'd by AMT Research.
Apple Special $175, Buy the Apple Z80 Developer at the same time as the "Chairman" and pay only $175 instead of $325.

APPLE Z80 DEVELOPER
Includes: 6502X ($150), MACRO II ($100), MACRO III ($150), QSAL ($200), QED ($150), LINKER ($80), DEBUG I ($80), DEBUG II ($100), ZDDT ($40) and BUSINESS BASIC ($200)
VALUE: $1250 — **NOW $325**
$175 when purchased with AMT "Chairman" Board

DISK FORMATS

When ordering software specify which disk format you would like.

CODE	DESCRIPTION
8SD	8" IBM 3740 Single Density (128 bytes/26 sectors/77 tracks)
8DD	8" Double Density (256 bytes/26 sectors/77 tracks)
8XD	8" CDL Extended Density (1024 bytes/8 sectcr/77 traceks = 616K)
5SD	5.25" Single Density (TRS80 Model I, Versafloppy I, Tarbell I)
5EP	5.25" Epson Double Density
5PC	5.25" IBM PC Double Density
5XE	5.25" Xerox 820 Single Density
5OS	5.25" Osborne Single Density
5ZA	5.25" Z80 Apple (Softcard compatible)

TPM INFO When ordering TPM I or II, in addition to Disk Format, please specify one of the following codes:

	CODE	DESCRIPTION
TPM I:	NSSD/H	North Star Single Density for Horizon I/O
	NSSD/Z	North Star Single Density for Zapple I/O
	NSDD/H	North Star Double Density for Horizon I/O
	NSDD/Z	North Star Double Density for Zapple I/O
	TRS80-I	TRS-80 Model I (4200H Offset)
	TRS80II	TRS-80 Model II
	VI8	Versafloppy I 8"
	VI5	Versafloppy I 5.25"
TPM-II:	VII8	Versafloppy II 8" (XD)
	VII5	Versafloppy II 5.25"
	TRS80II	TRS-80 Model II (XD)

Prices and Specifications subject to change without notice.
TPM, Z80, CP/M, TRS80 are trademarks of CDL, Zilog, DRI and Tandy respectively.

ORDERING INFORMATION:

VISA/MasterCard/C.O.D.
Call or Write With Ordering Information....

OEMS:

Many CDL products are available for licensing to OEM's. Write to Carl Galletti with your requirements.

Dealer Inquiries Invited.

For Phone Orders ONLY Call Toll Free...
1-(800) 458-3491 (Except Pa.)
Ask For Extension #15

For information and Tech Queries call
(609) 599-2146

Computer Design Labs

342 Columbus Avenue/Trenton, NJ 08629

Circle no. 15 on reader service card.

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
}
outdnhas()
{
        return(YES);
}
outsup()
{
        outxy(0,SCRNL1);
        sysdelay();             /*to control scroll rate */
        syscout(10);
}
outsdn()
{
        outxy(0,0);
        sysdelay();               /* to control scroll rate */
        syscout(27);              /* H19/H89 */
        syscout('L');
}
-------------------------------------------------------------------------------

/* ED7.C */

#include ed0.c
#include ed1.ccc
char pmtln[MAXLEN];
char pmtrfn[SYSFNMAX];            /* now both read and write filenames */
char pmtwfn[SYSFNMAX];
pmtmess(s1,s2) char *s1, *s2;
{
int x,y;
        x=outxset();
        y=outyset();
        outxy(0,0);
        outdelln();
        pmt1line();
        pmt1col(x);
        fmtsout(s1,outxset());
        fmtsout(s2,outxset());
        syscin();
        pmt1line();
        pmt1col(x);
        pmt1file(pmtrfn);       /* both read and write files */
        pmt2file(pmtwfn);
        pmt1mod(pmtln);
        outxy(x,y);
}
pmtmode(s) char *s;
{
int x,y;
        x=outxset();
        y=outyset();
        outxy(0,0);
        outdelln();
        pmt1line();
        pmt1col(x);
        pmt1file(pmtrfn);
        pmt2file(pmtwfn);       /* read and write files */
        pmt1mode(s);
        outxy(x,y);
}
pmtrfile(s) char *s;    /*changed: readfile */
{
int x,y;
        x=outxset();
        y=outyset();
```

(Continued on page 58, column 2)

```
        outxy(0,0);
        outdelln();
        pmt1line();
        pmt1col(x);
        pmt1file(s);
        pmt2file(pmtwfn);   /* add writefile */
        pmt1mode(pmtln);
        outxy(x,y);
}
pmtwfile(s) char *s;    /* new: writefile */
{
int x,y;
        x=outxset();
        y=outyset();
        outxy(0,0);
        outdelln();
        pmt1line();
        pmt1col(x);
        pmt1file(pmtrfn);
        pmt2file(s);
        pmt1mode(pmtln);
        outxy(x,y);
}
pmtedit()
{
        pmtmode("edit:    ");    /* add blanks to assure erasing "command" */
}
pmtline()
{
int x,y;
        x=outxset();
        y=outyset();
        outxy(0,0);
        outdelln();
        pmt1line();
        pmt1col(x);
        pmt1file(pmtrfn);        /* change: separate read and write files */
        pmt2file(pmtwfn);
        pmt1mode(pmtln);
        outxy(x,y);
}

/***** pmtcol() and pmtcmnd() unchanged *****/

pmt1mode(s) char *s;
{
int i;
        outxy(60,0);             /* change of column location */
        fmtsout(s,60);
        i=0;
        while (pmtln[i++]= *s++) {
                ;
        }
}
pmt1file(s) char *s;
{
int i;
        outxy(25,0);
        if (*s==EOS) {
                fmtsout("no rdfile",25);        /* change of message */
        }
        else {
                fmtsout(s,25);
        }
        i=0;
        while (pmtrfn[i++]= *s++) {
                ;
        }
```

(Continued on page 60, column 1)

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
}
pmt2file(s) char *s;     /* new: write out writefile name or 'no wrtfile' */
{
int i;
        outxy(40,0);
        if (*s==EOS) {
                fmtsout("no wrtfile",40);
        }
        else {
                fmtsout(s,40);
        }
        i=0;
        while (pmtwfn[i++]= *s++) {
                ;
        }
}

/***** pmtline1(), pmtcol1() changed in name to pmt1line(), pmt1col() *****/
--------------------------------------------------------------------------



/* ED8.C */

/* altered for Software Toolworks compiler with M80 option */
#include ed0.c
#include ed1.ccc
cpmcall(o,c) int o; char c;     /* new: cpmcall absent from original listing */
{
#asm
        POP     H       ;RETURN
        POP     D       ;CHAR C -- INPUT OR OUTPUT
        POP     B       ;INT 0 -- OPCODE
        PUSH    B
        PUSH    D
        PUSH    H       ;REPLACE RETURN
        CALL    5
        MOV     L,A
        MVI     H,0
/*      RET              Note: Software Tookworks compiler supplies RET */
#endasm
}       /* added */
syscstat()
{
int c;
        c=cpmcall(6,-1);
        if (c==0){              /* correction of original version */
                return(-1);
        }
        else {
                return(c);
        }
}
syscin()        /* extensively altered to recognize escape sequences */
{               /* from H19/H89 special keys */
int s,c;
        while ((c=cpmcall(6,-1))==0) {
                ;
        }
        if (c==ESC1) {
                while ((c=cpmcall(6,-1))==0) {
                        ;
                }
                if (c=='?') {
```

```
        else if (c==LF) {       /* map LF to CR for Software Toolworks  */
                return(CR);     /* compiler */
        }
        else {
                return(c);
        }
}
syspshch(c,file) char c; int file;
{
        if (c==CR) {            /* map CR to LF for Software Toolworks  */
                if (putc(LF,file)==-1) {        /* compiler */
                        error("disk write failed");
                        return(ERR);
                }
        }
        else if (putc(c,file)==-1) {
                error("disk write failed");
                return(ERR);
        }
        else {
                return(c);
        }
}

/***** syspopch() unchanged *****/


syschkfn(args) char *args;
{
        if (args[0] == EOS) {   /* add check for zero-length file name */
                message("no file name");
                return(ERR);
        }
        else {
                return(OK);
        }
}


/***** syscopfn() unchanged *****/


sysdnmov(n,d,s) int n,d,s;
{
/* extensively altered to allow LDDR for Z80 processors using parity check
   for processor type.  Also checks for zero-length move to avoid moving
   everything */
#asm
        POP     PSW     ;RETURN
        POP     H       ;SOURCE
        POP     D       ;DESTINATION
        POP     B       ;LENGTH
        PUSH    PSW     ;RESTORE RETURN
        MOV     A,C
        ORA     B
        JZ      SYSDN3
        MVI     A,2
        INR     A
        JPE     SYSDN1
        DB      0EDH,0B8H       ;LDDR INSTRUCTION FOR Z80
        JMP     SYSDN3
;
SYSDN1: MOV     A,B
        ORA     C
        JZ      SYSDN3
        MOV     A,M
        STAX    D
        DCX     H
```

```
		while ((c=cpmcall(6,-1))==0) {
			;
		}
/* "switch" and "case" construct can be altered to a sequence of   */
/* "else if" statements if compiler lacks "switch".                 */
/* Looks slow, but main body is executed only for escape sequences  */
		switch (c) {
			case 'a': return(UP1);
			case 'x': return(UP2);
			case 'r': return(DOWN2);
			case 't': return(LEFT1);
			case 'v': return(RIGHT1);
			case 'w': return(INS1);
			case 'M': return(ESC1);
			case 'y': return(DEL1);
			case 'p': return(GTOCH);
			case 's': return(ZAP1);
			case 'n': return(DTOCH);
			case 'u': return(HOME);
			default: continue;
		}
	}
	else {
		switch (c) {
			case 'S': return(DSCROL);
			case 'W': return(USCROL);
			case 'U': return(GOTO);
			case 'V': return(LSTRT);
			case 'Q': return(ABT1);
			case 'P': return(EDIT1);
			case 'T': return(LEND);
			case 'R': return(ESC1);
			case 'J': return(ERASE);
			default: return(c);
		}
	}
    }
    else {
	return(c);
    }
}


/***** CP/M VERSION 1.4 ROUTINES OMITTED *****/


/***** syscout() and syslout() unchanged *****/


sysend()
{
#asm
	LHLD	6
	DCX	H
/*	RET		Note: Software Tookworks compiler supplies RET */
#endasm
}	/* added */


/***** sysopen() and sysclose() unchanged *****/


sysrdch(file) int file;
{
int c;
	if ((c=getc(file))==-1) {
		return(EOF);
	}
```

(Continued on page 60, column 2)

```
	DCX	D
	DCX	B
	JMP	SYSDN1
;
SYSDN3:	POP	H
	PUSH	H
	PUSH	H
	PUSH	H
	PUSH	H
/*	RET		Note: Software Tookworks compiler supplies RET */
#endasm
}	/* added */
sysupmov (n,d,s) int n,d,s;

/* extensively altered to allow LDIR for Z80 processors using parity check
   for processor type.  Also checks for zero-length move to avoid moving
   everything */

{
#asm
	POP	PSW	;RETURN
	POP	H	;SOURCE
	POP	D	;DESTINATION
	POP	B	;LENGTH
	PUSH	PSW	;RESTORE RETURN
	MOV	A,C
	ORA	B
	JZ	SYSUP3
	MVI	A,2
	INR	A
	JPE	SYSUP1
	DB	0EDH,0B0H		;LDIR INSTRUCTION FOR Z80
	JMP	SYSUP3
;
SYSUP1:	MOV	A,B
	ORA	C
	JZ	SYSUP3
	MOV	A,M
	STAX	D
	INX	H
	INX	D
	DCX	B
	JMP	SYSUP1
;
SYSUP3:	POP	H
	PUSH	H
	PUSH	H
	PUSH	H
	PUSH	H
/*	RET		Note: Software Tookworks compiler supplies RET */
#endasm
}	/* added */

amatch (a,b,c) int a,b,c;
/* matches pattern at b to line at a
   all all question marks match
   if flag c is NO match is only at beginning of line.
   Returns column of match if match, otherwise -1
*/
{
#asm
	POP	PSW	;RETURN ADDRESS
	POP	B	;FLAG
	POP	D	;PATTERN ADDRESS
	POP	H	;LINE ADDRESS
	PUSH	H
	PUSH	D
	PUSH	B
```

(Continued on page 62, column 1)

Screen Editor Enhancements

(Listing continued, text begins on page 38)

```
        PUSH    PSW
        SHLD    LINAD
        XCHG
        SHLD    PATAD   ;PATAD IN HL, LINAD IN DE
        MVI     A,0
        ORA     C
        JNZ     MTCH1
        ORA     B
MTCH1:: PUSH    PSW     ;Z FLAG IS 'NO'
        LXI     B,0
        MOV     A,M
        CPI     0
        JZ      MTCH3   ;ZERO LENGTH MATCH STRING - NOMATCH
MTCH5:: CPI     '?'
        JZ      MTCH2   ;SUCCESS SO FAR
        LDAX    D
        CMP     M
        JZ      MTCH2
        CPI     0       ;END OF LINE
        JZ      MTCH3   ;NOMATCH
        POP     PSW
        PUSH    PSW
        JZ      MTCH3
        LHLD    LINAD
        INX     H
        INX     B
        SHLD    LINAD
        XCHG
        LHLD    PATAD
        MOV     A,M
        JMP     MTCH5
LINAD:: DS      2
PATAD:: DS      2
MTCH2:: INX     H
        INX     D
        MOV     A,M
        CPI     0
        JZ      MTCH4   ;TOTAL SUCCESS
        JMP     MTCH5
MTCH3:: LXI     B,-1
MTCH4:: POP     PSW
        MOV     H,B
        MOV     L,C
#endasm
}
sysdelay()
{
/* sets scrolling rate for scroll up or down - modify by
changing number loaded into D register - smaller value for slower scroll */
#asm
        ANA     A
        LXI     D,10
        LXI     H,0
DOMOR:: DAD     D
        JNC     DOMOR
#endasm
}

--------------------------------------------------------------------------------


/* ED9.C */

#include ed0.c
#include ed1.ccc
```

```
                return;
        }
        if (bufatbot()) {
                outdeol();
        }
        else {
                c=fmtsout(bufp,0);                      /* c= added */
                if (c<SCRNW1) {                          /* now allows full screen */
                outdeol();
                }
        }
}

/****** bufext() unchanged *****/


/* bufstrch is bufext modifed to allow entering beyond buffer with warning */
bufstrch(length) int length;
{
        if ((bufpmax+length)>=bufend) {
                bufdnmov(bufp,bufpmax-1,length);
                bufpmax=bufpmax+length;
                message("Caution: main buffer is full");
                return(ERR);
        }
        bufdnmov(bufp,bufpmax-1,length);
        bufpmax=bufpmax+length;
        return(OK);
}


buflength(first,n) int first, n;    /* new: returns length of n lines from  */
{                                   /* first line specified */
char *start;
int i,len,line;
        if (n<=0) {
                len=0;
                return(len);
        }
        if (bufgo(first)==ERR) {
                len=0;
                return(len);
        }
        start=bufp;
        i=1;
        line = first;
        while (i++<=n) {
                if (bufgo(++line)==ERR) {
                        len=0;
                        return(len);
                }
        }
        len=bufp-start;
        return(len);
}


bufcopy(from,to,n) int from,to,n;       /* new: copies "n" lines from "from"  */
{                                       /*      to "to" */
int length;
char *source,*dest;
        if (from==to){
                return;
        }
        if ((length=buflength(from,n))==0){
                return;
        }
        if (bufgo(from)==ERR){
                return;
        }
        source=bufp;
        if (bufgo(to)==ERR){
                return;
```

```
/***** ED9.C functions unchanged *****/
------------------------------------------------------------------------


/* ED10.C */

#include b:ed0.c
#include b:ed1.ccc
int bufcflag;
char *bufp;
char *bufpmax;
char *bufend;
int bufline;
int bufmaxln;
/* mbuffer is external for Software Toolworks compiler to force it beyond
   code sections */
extern char mbuffer[1];
bufnew()
{
        bufp=bufpmax=mbuffer+1;
/* need more room for Software Toolworks stack and buffers */
        bufend=sysend()-1500;
        bufline=1;
        bufmaxln=0;
        mbuffer[0]=CR;
        bufcflag=NO;
}

/***** bufln(),bufchg(),bufsaved(),buffree(),bufgo(),bufup(), bufdn(),


                bufins() unchanged  *****/

buflins(p,n) char *p; int n;      /* added: a full buffer does not truncate */
{                                 /*   line, but issues warning and returns  */
int k, x;                         /*   ERR - allows editing of large files   */
        x = 0;                    /*   in pieces */
        if(bufstrch(n+1)==ERR) {
                x = 1;
        }
        k=0;
        while (k<n) {
                *(bufp+k)= *(p+k);
                k++;
        }
        *(bufp+k)=CR;
        bufmaxln++;
        if ((n==0)&(bufnrbot())) {
                ;
        }
        else {
                bufcflag=YES;
        }
        if (x == 0) {
                return(OK);
        }
        else {
                return(ERR);
        }
}

/***** bufdel(), bufrepl(), bufgetln(), bufdnmov(), bufupmov(),
       bufatbot(), bufnrbot(), bufattop(), bufout() unchanged *****/

buflnout(line) int line;
{
int c;
        if (bufgo(line)==ERR) {
                fmtsout("disk error: line deleted",0);
                outdeol();
```

(Continued on page 62, column 2)

```
        }
        dest=bufp;
        if ((bufpmax+length)>=bufend) {
                message ("insufficient room in buffer for copy or move");
                return;
        }
        if (bufext(length)==ERR){
                return;
        }
        bufgo(1);
        bufmaxln=bufmaxln+n;
        if (from<to) {
                sysupmov(length,dest,source);
        }
        else {
                sysupmov(length,dest,source+length);
        }
}
------------------------------------------------------------------------


;MBUFFER.MAC (REQUIRES M80 TO ASSEMBLE) -- MUST BE VERY LAST MODULE LINKED

        ENTRY MBUFFER
MBUFFER::       DS      0
        END
```

End Listing

Shifts and Rotations on the Z80

The shift and rotation commands on the Z80 are powerful commands. They allow simple division and multiplication. Along with the BIT command, they provide powerful control over the eight bits in each byte. But for all the brilliant applications, how does one know which specific type of shift or rotation one needs? Since I could not find an explanation of the discrete differences between these commands, I compiled the following with a little research and experimentation. Hopefully it will prove helpful when employing these powerful commands in any future assembly/machine language endeavors. Note that whenever the command is "xxx r," the "r" refers to any single, unpaired Z80 register other than "F" and a couple of other things. Thus, r can be A, B, C, D, E, H, L, (HL), (IX+d), or (IY+d).

(1) RR r – Rotate Right thru Carry. This command moves each bit in the chosen register to the right. The bit in position 0 goes to the carry flag, and the carry flag status is placed into bit position 7. This command is valuable because it allows for examining each individual bit's status. Each time RR r is performed, the next bit is placed in the carry, allowing appropriate action – e.g., "JR C,BITON." An example of this command may be found in the program in Figure 1 at right.

(2) RLC r – Rotate Left Circular. This moves each bit in the specified register to the left. The bit in position 7 of the register is put in position 0. The use of this rotate is probably less common. It can multiply a number by two if the result is less than 255. If the result is greater than 255, the status of bit 8 (256th's place, which is really the ninth bit and so normally not there) will be stored in bit position 0. Bit position 0 will hence equal 1 if the result is greater than 255, and 0 if the result is less than or equal to 255. Figure 2 at right shows a program using RLC. Notice that it uses the "A" register but it could use any of the "r" values.

(3) RL r – Rotate Left thru Carry. This moves each bit to the left, but the bit in position 7 goes to the carry flag and the carry flag status is put into bit 0. This command can be used like RR r, except that this rotates the other direction. If it is used for multiplication and the result is greater than 255, the carry flag will be set.

(4) RRC r – Rotate Right Circular. This moves each bit to the right. The bit in position 0 of the byte is put into position 7. This command works like RLC r except it rotates the other direction and hence it divides.

(5) SLA r – Shift Left Arithmetic. With this command all of the bits are shifted to the left. The status of bit 0 is always 0 after this shift is used, and bit 7 is lost. This is best described in Table 1 on page 66. This command has a somewhat more useful multiply ability for most applications. It multiplies by two and leaves 0 at bit position 0. After multiplying any binary number by two, bit position 0 should equal 0 because the answer is even.

(6) SRA r – Shift Right Arithmetic. All of the bits are shifted to the right. The status of bit 7 remains unchanged, but bit 0 is lost. Although this command can be used for division, the main problem is that it leaves bit 7 on and hence it will give an inaccurate result if the number to be divided is greater than 127. There may be some practical reason for leaving bit 7 on, but I am unaware of it.

(7) SRL r – Shift Right Logical. This command is very similar to SRA r. All of the bits are again shifted to the right, and bit 0 is lost. But with this command, the status of bit 7 is always set to 0 after a shift. This makes this command far more practical for division than SRAr. Unlike SRAr, which will only give an accurate result if the number being divided is less than 128, this will give an accurate answer if the number to be divided is less than or equal to 255 (which will always be the case if it is being stored in an 8-bit register).

Table 1 shows how each of the commands operates. Each one is performed several times on a number to provide easy visualization of the command's results.

DDJ

(Table 1. on page 66)


```
; THIS PROGRAM WILL PRINT THE BINARY REPRESENTATION
; OF THE 'A' REGISTER ON A TRS-80 SCREEN AT
; SCREEN LOCATION 3C00H (15360 DECIMAL).
;
PRINT   LD      B,8
        LD      HL,3C07H
LOOP    RRA
        LD      A,48
        JR      NC,ZERO
        LD      A,49
ZERO    LD      (HL),A
        DEC     HL
        DJNZ    LOOP
        RET
```

Figure 1.

```
; PROGRAM TO MULTIPLY 'A' BY 2. PRODUCT MUST BE
; LESS THAN OR EQUAL TO 128.
; THE COMMAND RLCA DOES SAME THING AS RLC  A
;
MULT2   RLC     A
        RES     0,A
        RET
```

Figure 2.

by Ron Goodman

Ron Goodman, 12702 Emelita Street, North Hollywood, CA 91607.

RR	THE BYTE	CARRY	RRC	THE BYTE	SRA	THE BYTE	SRL	THE BYTE
	11111111	0		00000001		10100000		10100000
	01111111	1		10000000		11010000		01010000
	10111111	1		01000000		11101000		00101000
	11011111	1		00100000		11110100		00010100
	11101111	1		00010000		11111010		00001010
	11110111	1		00001000		11111101		00000101
	11111011	1		00000100		11111110		00000010
	11111101	1		00000010		11111111		00000001
	11111110	1		00000001		11111111		00000000
	11111111	0		10000000		11111111		

RL	THE BYTE	CARRY	RLC	THE BYTE	SLA	THE BYTE
	00000001	0		00000001		11111111
	00000010	0		00000010		11111110
	00000100	0		00000100		11111100
	00001000	0		00001000		11111000
	00010000	0		00010000		11110000
	00100000	0		00100000		11100000
	01000000	0		01000000		11000000
	10000000	0		10000000		10000000
	00000000	1		00000001		00000000
	00000001	0		00000010		00000000

Table 1.

SBC, TSX and TXS
Instructions of the 6800 and 6502

The 6800 and 6502 microprocessors have very similar architectures and instruction sets. The common instructions of these two microprocessors have been listed elsewhere[1,2] and the purpose of this note is to draw attention to three common instructions which have subtle but important differences in their operation on the two MPUs. These are:

SBC	Subtract with carry (on the 6800 the mnemonic is followed by A or B indicating the accumulator concerned)
TSX	Transfer stack pointer to index register
TXS	Transfer index register to stack pointer

SBC

The differences in the carry-bit operation for the subtract instruction of the 1802 and Z80 microprocessors have been pointed out by Merrin.[3] Likewise, the differences in SBC between the 6800 and 6502 are due to the different ways in which the two MPUs utilize the carry bit of their status registers. Though SBC is nominally described as subtract with carry, a better description would be subtract with borrow.

The add with carry instruction, ADC, is also common to the 6800 and 6502, and functions identically on them. The operation can be represented as

$$A + M + C$$

where A is the contents of the accumulator, M the operand, and C the carry bit of the status register. The result is put into the accumulator and the resulting carry bit is stored in C. The resulting carry bit will be 1 only if the result exceeds 255. In the case of the 6502, ADC is the only addition instruction, and if an add without carry is required, the carry bit must be cleared first by preceding ADC with CLC (clear carry bit).

To understand the SBC instruction better, let us assume that the status registers of both MPUs have another status bit called the borrow bit. The operation of the SBC instruction can then be represented as

$$A - M - B$$

by B.T.G. Tan

B. T. G. Tan, Department of Physics, National University of Singapore.

where B is the borrow bit. B, as well as M, is subtracted from A, and the result is put into the accumulator. The borrow bit resulting from the operation will be stored in B. The resulting borrow bit will be 1 only if the absolute value of (M + B) is greater than that of A. The result of the operation goes into the accumulator. The function of B is to facilitate multi-byte subtraction when borrow bits from one 8-bit subtraction have to be utilized in the subtraction for the next higher 8-bits.

If the subtraction without borrow is required, the borrow bit must be cleared before SBC. We can invent two pseudo-instructions

SEB	Set borrow bit
CLB	Clear borrow bit

to enable us to manipulate B, analogously to SEC and CLC for C. Hence we must precede SBC with CLB for a subtract without borrow.

In the 6800 and 6502 MPUs, there is no separate borrow bit in their status registers. The carry bit C does double duty as a borrow bit, so that it is better described as a carry/borrow bit. In the 6800, the borrow bit is identical with the carry bit, so that the SBC operation can be represented as

$$A - M - C$$

and the borrow bit generated is directly put into C. The instructions SEB and CLB correspond to SEC and CLC respectively.

In the 6502, the borrow bit is the inverse of the carry bit, so that SBC is

$$A - M - \overline{C}$$

and the borrow bit generated is inverted before being put into C. Hence the SEB instruction corresponds to CLC and CLB to SEC. For a subtraction without borrow, SBC would have to be preceded by SEC.

The difference is thus best understood by considering the borrow bit B as entirely separate from C and then taking $B = C$ for the 6800 and $B = \overline{C}$ for the 6502. The implementation of the borrow bit for the 6502 is different from the 6800 in order that arithmetic operations on the 6502 would give more consistent results for C.

For example, the arithmetic operations 5 - 3 and 5 + (-3), using the two's complement in the second operation, would lead to different values for C in the case of the 6800. These operations would give the same values for C in the case of the 6502, since its borrow bit is the inverse of C. In Table 1 (below), the two operations are listed for each MPU. showing the corresponding values of the accumulator and the carry bit after each instruction. These operations were verified on SWTPC 6800 and Apple II microcomputers. The 6809, which is the upgraded version of the 6800, appears to implement the borrow bit in a similar manner to the 6800, that is with the borrow bit the same as the carry bit.

TSX and TXS

The transfer instructions between the stack pointer SP and index register X, TSX and TXS are common to both 6800 and 6502 but have important differences. In both MPUs, the current value of SP points to the location one address below the bottom of the stack. These registers are 16-bits long in the 6800, and 8-bits long in the 6502.

In the 6800, TSX increments the contents of SP by 1 and then transfers them to X. The reason for this appears to be to make X point to the actual bottom

6800	ACCA	C	B=C	6502	ACC	C	$B=\overline{C}$
LDAA #$05	05	D	D	LDA #$05	05	D	D
CLC	05	0	0	SEC	05	1	0
SBCA #$03	02	0	0	SBC #$03	02	1	0
LDAA #$05	05	D	D	LDA #$05	05	D	D
CLC	05	0	0	CLC	05	0	1
ADCA #$FD	02	1	1	ADC #$FD	02	1	0

Table 1.
5 minus 3 subtraction operations for 6800 and 6502. D indicates don't care state.

of the stack. This is presumably for cases when indexing operations on the stack are to be carried out. Conversely, TXS decrements the contents of X by 1 before transferring them to SP, so that if X pointed to the bottom of the stack, SP would then point to the next location below the bottom.

The 6502's TSX and TXS instructions do not perform the increment and decrement operations; they are just straightforward data transfer instructions. Why is this so? SP is an 8-bit register, and its contents ZZ are always interpreted as pointing to the page 1 of memory, $01ZZ. The effective address in indexed addressing is found by adding the 8-bit contents of X to the operand. In the zero-page indexed mode, the effective address is in page zero; in the absolute indexed mode, the page of the effective address depends on the 16-bit operand.

Because of these complications, the TSX and TXS instructions of the 6502 do not appear to be intended to facilitate indexed operations on the stack, as the transfers are effected without incrementing and decrementing. The primary function of TSX and TXS appears to be to effect data transfer to and from SP via X. This is in fact the only way to do so, since the 6502 lacks LDS (load to stack pointer) and STS (store from stack pointer) instructions. These instructions are part of the 6800's instruction set.

The 6809 MPU, which is the upgraded version of the 6800, has the instructions TFR S,X and TFR X,S which are equivalent to TSX and TXS, respectively. In the 6809, the stack pointer points to the bottom of the stack, and not to the location below the bottom as in the case of the 6800. Thus, the two transfer instructions do not perform an increment or decrement before the transfer, and are more like their 6502 equivalents, but for a quite different reason.

Conclusions

The 6800 and 6502 MPUs are so similar that it is worth looking at both together when studying one of them. I have taken this approach in my microprocessor teaching, using a MPU model consisting of the common features of the two MPUs.[2] This MPU model has one 8-bit accumulator, one 8-bit index register, one 16-bit program counter, one 8-bit stack pointer, and one 8-bit status register. It has the same six addressing modes of the 6800, except that indexed addressing is restricted to the zero page only. The instruction set is made up of the 43 instructions common to both 6800 and

(Continued on page 85)

ZAS Z8000
Software Development Package
By Western Wares, P.O. Box C, Norwood, CO 81423
CP/M Version $395
ISIS Version $495
Reviewed by Terry R. Dettmann
Circle No. 151 on Reader Service Card.

What is there to say about an assembler? It either assembles good code or it doesn't. ZAS assembles functioning Z8000 code (consistent with your own ability to design such code in the first place!).

Is there any more to say? Should a review really continue after this point? Well in some cases I'd say no it shouldn't, but in this case there is more to the package that is important to know about. More that the package can do. What more should I say? Well, let's look and see just what you get when you get the assembler package.

First there's the ZAS assembler itself. It is a "full-featured, relocatable cross assembler for the Z8000." Its instruction mnemonics are compatible with those developed by Zilog. Comparing it to Matosian's book on the Z8000 (SYBEX), it didn't always agree on how some things should be done, but it was easy to learn.

ZAS also includes the "extended" instructions needed for some Z8000 units. It wasn't hard to use. It generated good code for the Z8000, but has only really been tested on limited development systems. I haven't tried to code a large project.

ZAS is a two-pass assembler. It has all the standard things you expect to find in a full-featured assembler such as conditional assembly, named sections, segmented and non-segmented addressing support, nested include files, up-to-64-character symbol names, and standard directives and pseudo-ops.

After the assembler comes the ZLK task builder. ZLK is a flexible linking program that allows complicated processes to be set up as overlays and so forth, while allowing the programmer complete control over memory placement of the code modules.

ZLD is an absolute file loader program. Its basic purpose is to load a Z8000 module under CP/M or ISIS so that it can be executed with the ZEX run-time monitor. ZEX allows the user's system to provide run-time support as needed for a Z8000 task. ZEX allows bus-switching between 8080-type CPUs and Z8000 CPUs, as on the IEEE 696 (S-100) bus or the Intel Multibus. It even allows the program to make CP/M or ISIS calls as needed.

In this kind of mode, the Z8000 CPU is acting as if it were a temporary bus master device and the 8080 CPU is the bus master. All I/O is handled by the bus master (the 8080) through the CP/M system.

ZEX is configured by the user to the particular environment. Detailed instructions are provided to allow proper configuration by anyone with reasonable technical knowledge. This isn't a system for the novice Z8000 programmer, but it is a useful tool for serious system development.

Editor's note: Just before press time, we were informed that version 2.0 of the ZAS Z-8000 package had been released. Some of the enhancements will be of particular interest to those familiar with older versions of ZAS. The following new features were added in the new version, which was received too late for review in this issue: new macro pseudo-operations (MACRO, IRP, IRPC, EXITM, REPT, ENDM, LOCAL), other new pseudo-ops (DZ – define zero, DA – define address), new directives ($BASE – set default number base; $LIST, MACON, MACOFF – control macro expansion listing), new comparison opertors taking string or numerical arguments, extended instruction enhancement, more symbol table space and faster searching of tables, more flexibility for input, command-line switches specified by - or /, and ZAS and ZLK now use the same command-line syntax.

6809 Cross Assembler
Available from Avocet Systems Inc., 804 South State Street, Dover, DE 19901
$200
Reviewed by Terry R. Dettmann
Circle No. 153 on Reader Service Card.

It wasn't long ago that I came up with an acute need for a 6809 assembler. Not just a small one either. Further, I was lazy and I wanted full-screen editing ability, flexible printing, and large capacity. In short, I wanted a large system! In fact, all I had was a small system.

It was then that the Avocet ad in some of the computer magazines caught my eye. As a CP/M cross assembler for the 6809, my dreams were fulfilled. I could have all the powerful CP/M tools I wanted and still be able to put things together for a 6809. I could document my code in as much detail as I wanted. I could modularize as much as I wanted. I could work with 6809 code in a convenient way.

It wasn't long before I started putting together some 6809 software including some communications packages and a simple Forth system. I found out just how lazy I had become when I tried to go back to the little 6809 assembler I had. It was good, but just wasn't made for the kind of programming I was doing.

Avocet produces a whole line of cross assemblers which include assemblers for the following processors: 6805, 6809, RCA1802 (COSMAC), Intel MCS-48 & UPI-41 families, Intel 8051, MOS 6502 & 650X family, Motorola 6800/6801 family, NEC 7500 family, Fairchild F8, MOSTEK F8/3870 family, National COP400, and the Zilog Z8 family. Quite a list.

No assembler is worth its salt, though, unless it can produce good code. The Avocet 6809 assembler does as far as I have gone with it.

So far, I haven't run across any errors not traceable to my own programming errors. If the assembler would only assemble what I want instead of what I type. But the point is that it does do what I type!

The basic assember syntax is roughly the same as for a standard 8080 assembler including standard pseudo-ops such as DW, DB, etc. The write-up with the system says that all the assemblers stay with these conventions. I'm not sure I agree that this is the best though I'm sure it makes development and maintenance easier for the Avocet people.

Personally, I would rather have seen the pseudo-ops closer to those used by the standard system for the 6809, but I can live with anything as long as it produces the code I want.

The assembler produces code in one of two output formats, either Intel Hex format or MIKBUG format. I use the Intel Hex format mostly out of habit.

This is a nice assembler with reasonable documentation for the experienced programmer. It carries out the job I want it to do; after all, isn't that what it should do?

DDJ

BCPL — The Language and Its Compiler
by Martin Richards
and Colin Whitby-Strevens
Cambridge University Press
173 pages, $10.95
Reviewed by George W. Jolly

They say you can't tell a book by its cover, but sometimes I'm not so sure. The cover of *BCPL* is a gentle interplay of two colors, blue and green, producing results you might think would require three or more inks. *BCPL* is a language very much in that style, favoring careful design over expensive implementation.

BCPL (Basic CPL) is a systems programming language designed by Martin Richards in 1967. The language, developed mostly over the subsequent five years, has been installed on at least twenty-five different computer systems. It has been used for applications including operating systems, compilers, editors, databases — in other words, for a broad spectrum of systems work.

In *The C Programming Language*, Kernighan and Ritchie credit the language BCPL for many of C's most important ideas. BCPL influenced the language B written for UNIX by Ken Thompson in 1970, and B passed this influence on to C. However, BCPL and C are quite different, perhaps most importantly in the handling of data types.

Many modern languages have a variety of data types built into the compiler. Such types might be integer, floating point, character, pointer, and so on. BCPL, on the other hand, has only one type, the machine word. The compiler wastes no time identifying the legality of a construct for the data items involved, because all data items are alike. In BCPL, address variables, or pointers, are easy to use because indirect addressing is a fundamental operator. Any data item may contain an address.

One interesting result of this inexpensive philosophy is that the elements of an array need not all be the same "type." (Try having integers, characters and pointers in the same array in standard FORTRAN!) Other languages do provide this, but as an extra feature. In BCPL it is a consequence of the underlying design. (Funny how FORTH comes to mind at this point.)

Example BCPL routines illustrate the power of the language. One example given is a formatted output procedure accepting a format string (analogous to a FORTRAN format) as the first argument, and the data list as subsequent arguments. A procedure receives its arguments as an array of addresses. It is a simple matter to scan this array with a pointer variable, accessing each parameter in turn. Another pointer variable holds the address of an appropriate editing procedure as indicated by the format string. This is a simple procedure with elegant results.

Other example programs include an interactive debugging program and the lexical/syntax scanner from the BCPL compiler. The latter is discussed in detail, with a page of program facing a page of explanatory text. The applicative expression tree produced by the scanner is an interesting topic in itself.

Another interesting topic of this book is language portability. The authors illustrate a method of bootstrapping the compiler onto a new system by generating a simplified pseudo-code on an existing installation. They also discuss common library procedures that exist in most BCPL installations. These topics may be of interest to implementors of other languages.

In summary, *BCPL* is a well-written book serving several possible needs. It describes an influential (and useful) computer language, illustrates an elegant programming philosophy, and gives practical examples of compiler techniques. The authors' straightforward explanations, like the cover design, complement the elegant simpilicty of BCPL.

User's Guidebook to Digital CMOS
Integrated Circuits
by Eugene R. Hnatek
McGraw-Hill
339 pages, $29.00
Reviewed by Eunice B. Stetson

User's Guide to Digital CMOS Integrated Circuits by Eugene R. Hnatek is not a book for the uninitiated. It is most likely to appeal to design engineers and sophisticated hobbyists. It assumes some knowledge of semiconductors, small- and medium-scale logic circuits, conversions between digital and analog signals, and computer hardware. Very little knowledge of mathematics beyond the ability to read complex graphs is assumed. The book is very practical in orientation. Many applications, with schematics, manufacturers' names, and part numbers, are included.

The author makes clear the electrical characteristics of CMOS as compared with other IC logic families. He does not pretend to discuss all CMOS integrated circuits but concentrates instead on medium- and large-scale IC's developed since 1977.

The author uses many schematics and graphs and some charts and timing diagrams to explain the exciting and promising developments in CMOS and SMOS/SOS (silicon on sapphire) technology and applications in telecommunications, computers and process controllers. He envisions many new uses for CMOS large-scale integration, particularly in the interfacing of analog and digital signals on a single IC. Currently, applications of CMOS microprocessors are concentrated in the automotive, games, and TV industries rather than in the computer field.

The author does an outstanding job of making his graphs understandable with labelling and notes on the graphs. Circuit diagrams for MSI and LSI integrated circuits are arranged in a logical sequence with explanation.

In discussing flip-flops, memory cells, and shift registers, the author uses schematics with FET symbols rather than logic symbols. Later he treats counters, timers, multiplexers, comparators, decoders, VARTs, and microprocessors as black boxes. As a teacher I would have preferred a more orderly progression from FETs to logic symbols to flip-flops to shift registers, multiplexers, etc. as the basic building blocks of which circuits are composed. However, in that this book is intended as a reference book and not a text book, this should not be a problem. As with all books which use manufacturers' data and diagrams, there are differences in symbols and conventions with which the reader is used to coping.

Generally, the author's explanations are clear, though as with most authors what follows "it can be seen that . . ." is not always obvious. The author clearly knows his subject well. At times he makes beautifully clarifying statements. This book, compared with others in its field, is clearly written, supplemented with many excellent pictorials, and does an authoritative job in the areas covered.

(Continued on page 72)

Book Reviews

(Continued from page 70)

Highfalutin' Computin' with Bob Orrfelt On your Timex Sinclair 1000™ Computer
By **Bob Orrfelt, 3436 Bay Road, Redwood City, CA 94063**
$9.95, 116 pages
Reviewed by **David S. Lacey**

Bob Orrfelt's book on the TS-1000 brings back an occasion when I submitted a somewhat hastily put together paper in a college English course, and got it back with a fairly good grade, but with a notation from the professor, "This is so good I wish it were better." He explained that this was his way of saying he liked my ideas, but felt the presentation just wasn't up to the content. If I had taken more pains in the preparation, I could have had a strong "A" instead of a marginal "B+." I think something very similar applies to *Highfalutin' Computin'*.

The book has lots to offer – things to instruct or amuse readers with a variety of viewpoints and interests. It has much for readers who are just getting an introduction to computers via the TS-1000. It also has considerable material for others already more versed in computers and their technology but interested in the particulars of the TS-1000 which, at under $100, is so attractive for experimenting. Unfortunately, the book's diversity is one of the underlying faults that brings the "grade" down. Mr. Orrfelt does not seem to have settled very firmly in his own mind on a particular audience for his effort. He shifts, sometimes rather abruptly, from one level of technical sophistication to another and may leave some less knowledgeable readers feeling rather baffled from time to time. I believe, from my own experience, that if readers persist through areas that seem to have them out of their depth, they will reach places in which they feel at ease again, and can profit from doing so. However, the book would have been more satisfactory if it had either stuck to one level, or built steadily from the simplest to the most sophisticated levels.

There is definitely more technical information in this book than others I have encountered on the TS-1000 or its predecessors, the Sinclair ZX80 and ZX81. This includes some fairly elementary but very practical hardware coverage. For those who can't abide the membrane keyboard, there are instructions on how to get the case open and connect a "real" keyboard in place of the membrane one. There is a circuit for obtaining a composite video signal to drive a monitor (instead of the modulated RF carrier normally output for connection to the antenna terminals of a TV receiver). Also presented is the complete circuit of an automatic control for a cassette recorder (the normal storage medium for the TS-1000). This information includes interfacing, both hardware and software, to make it entirely workable. They are all very simple but practical circuits, and can be regarded as introductions to some of the explorable possibilities of the computer.

There is a chapter that comprises a fairly detailed explanation of the internal organization of the computer, its operating cycles and its files. Programs for printing out the contents of the files are given, and the whole approach is designed to help enhance the reader's ability to write more compact, faster-running programs in BASIC.

There are a number of program listings of varying complexity throughout the book. These include games (a simple arcade-type game, a demonstration of moving graphics, a crap game, and blackjack); "useful" programs (length, weight, and volume converters, etc.); and what may be called "general interest" programs (such as two clock programs – one a dial face and one a digital clock with large numerals). None of these listings are presented for their own sake, rather as part of a tutorial approach that starts with a brief explanation of Sinclair BASIC. Throughout the rest of the book, the programs are explained in considerable detail regarding how desired results are obtained and with repeated emphasis on compacting – an essential when working with only 2K of RAM. The programming material culminates in a glimpse at machine code programming, though more than a glimpse is beyond the scope of this book.

Unfortunately, the subject of programs brings me to the second important fault which I find in Orrfelt's effort. There are several errors in programs. True, I found no true programming errors. Rather, they are typographical in nature, but they make some of the programs run faultily, or not at all. I can say that I, who am not much beyond the novice level, have been able to correct all the errors I found without too much difficulty. I may have learned valuable lessons in the essential art of debugging, but that isn't why they are there. Programs that do not run right do not belong in a "grade A" computer book.

Despite the faults mentioned, and weighing them against the virtues of a considerable amount of good technical information and explanation, I feel that readers who are in the earlier stages of learning the TS-1000 can, through a little persistence, gain their full money's worth from *Highfalutin' Computin'*.

We are informed that the book now comes with an addendum which corrects typographical errors such as those noted by the reviewer. – Ed.

DDJ

CP/M EXCHANGE

by Robert Blum

Digital Research created a stir when they announced CP/M Plus (alias CP/M 3) several months ago. Preliminary speculation on the system's content ranged from a simple maintenance release to full concurrent operation.

If you consider yourself a hacker, a unique individual who treats inadequacy as an opportunity, CP/M Plus will be bittersweet. Many of its performance options are and have been available in the public domain for years. If nothing else, CP/M Plus verifies that when hackers speak, the manufacturers listen. It just takes them a little while to react.

Even though I had hoped for more, the official release is anything but disappointing. It is chock-full of new features geared to performance and versatility, and is a major departure from the CP/M we have grown to love and respect. DR views CP/M Plus as a new product and it will be maintained and marketed separately. At least for now, it will not obsolete CP/M 2.2.

Over the next few months I will be covering various aspects of CP/M Plus: does it work as advertised; how difficult is it to implement; is it worth a $350 investment? Getting to know a new product cannot be effectively accomplished alone. Share your experiences and thoughts with the rest of us by dropping me a note or calling me in the evening. Your input will be appreciated by one and all.

Before getting into this month's topic, I need some help. Is there a patch for CP/M 2.2's CCP that will automatically search user area 0 for a .COM file that was not found in the current user area? If you have one, please send it along or point me to where I can find it.

After a long wait, CP/M Plus is now being shipped. By the time this column is published, it will probably be available through mail order and possibly from a few manufacturers. This estimate may be a little optimistic because implementation of all CP/M Plus's new features requires considerable attention to both hardware and software.

Even though I have not received my own distribution version of CP/M Plus, I have been able to gather up two of the three manuals and spend a few hours using it. What a pleasant surprise! The documentation has been completely rewritten and downsized to wedgy format. Each topic is thoroughly discussed in plain English, with only minimal use of buzz words, and is accompanied by plenty of examples.

Sitting at the keyboard for the first time was disappointing. I had hoped for concurrent operation, or at least a built-in print spooler. CP/M Plus remains single user and still prompts with the familiar A>. I soon learned, however, that all other aspects of the system are new, and verified that this is a completely new product. After only a few minutes at the keyboard I was comfortable with the operator interface and swear by the additional line-editing features. For those who are new to CP/M, the HELP command, which displays summarized information on all the commands, will undoubtedly prove invaluable.

The actual performance of the system, and which features can be implemented, is predicated on how the system is generated. As a replacement for CP/M 2.2 in a standard 64K or less environment (nonbanked), CP/M Plus offers less than 96K of banked memory when available.

Command-line editing on nonbanked systems remains the same as CP/M 2.2, but is greatly improved on banked systems. In the banked environment it is no longer necessary to retype a command if a mistake is made. Simply move the cursor to the error and correct it by inserting or deleting characters. Even after hitting return, commands can be redisplayed and edited provided a transient program was not executed. Refer to Table 1 on page 76 for a complete comparison of the available editing functions.

A host of new commands has been added to those that are carried over from older versions of CP/M. DIRectory, ERAse, REName, TYPe, and USEr are improved while DIRSys has been added to display system files which previously required the use of STAT. Depending on command-line options, transient programs may be automatically loaded to assist the built-in commands of the CCP. I have summarized the built-in commands in Table 2 (page 76). The descriptions used for each command are excerpts from DR's manuals. I do this so that you can sample the flavor of the new documentation.

Enhanced command editing is only a part of the beauty of CP/M Plus. Many new utilities have been added and those that were retained from older versions are now more useful. Also included is the entire RMAC program development package which previously cost $200. Refer to Table 3 (page 78) for a summary of the utility programs.

Before generating the system, you are faced with a dilemma. The minimum system configuration has been increased to 32K and CP/M Plus requires 8.5K. By the time you add in the size of your BIOS a great deal of memory has been taken up, even when a full 64K is available. To make full use of the system requires at least 96K of bank-select memory. Through the use of this additional memory, it is entirely possible to have a 62K TPA because only a minimum BIOS and BDOS are resident in the application bank. The real controlling logic is placed in the other bank along with directory hash tables and sector buffers. Because of this, I would anticipate the introduction of more 8-bit machines with 256K of memory in the not-too-distant future.

Disk space reserved for the system tracks has been reduced by placing the operating system components into regular disk files. This will be especially important for those systems that require very large BIOSs. When booting the system, a small loader from the system tracks is read into memory, which then completes the loading process. In light of more common use of hard disks, individual file size has been increased to 33 megabytes and disk capacities of 512 megabytes are possible.

Generation of the system is handled by a transient program called GENCPM which combines the various system components with your BIOS. The generation process is remarkably similar to that used with MP/M, as are most other features of CP/M Plus.

CP/M Plus is not for everyone. Its initial cost is high and it requires expanded hardware resources if it is to be used to its fullest extent. But the costs associated with its implementation will not seem nearly as high if you need all the development tools which are included. Functionalism is the keyword to describing this new offering from Digital Research.

Next month I will continue with the internals of CP/M Plus and any experiences that I hear from you. You can reach me by phone at (404) 449-8948. DDJ

(Tables I and II on page 76)

(Table III on page 78)

TABLE I

Nonbanked Command Editing

Ctrl-E Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing your previous input.

Ctrl-H Deletes a character and moves the cursor left one character position.

Ctrl-I Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the tab key.

Ctrl-J Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a Ctrl-M.

Ctrl-M Sends the command line to CP/M 3 and returns the cursor to the left of the current line. Has the same effect as a RETURN or a Ctrl-J.

Ctrl-R Places a # at the current cursor location, moves the cursor to the next line, and displays any partial command you typed so far.

Ctrl-U Discards all characters in the command line, places a # at the current cursor position, and moves the cursor to the next command line.

Ctrl-X Discards all the characters in the command line, and moves the cursor to the beginning of the current line.

Banked Command Editing

Ctrl-A Moves the cursor one character to the left.

Ctrl-B Moves the cursor to the beginning of the command line without having any effect on the contents of the line. If the cursor is at the beginning, Ctrl-B moves it to the end of the line.

Ctrl-E Forces a physical carriage return but does not send the command line to CP/M 3. Moves the cursor to the beginning of the next line without erasing the previous input.

Ctrl-F Moves the cursor one character to the right.

Ctrl-G Deletes the character indicated by the cursor. The cursor does not move.

Ctrl-H Deletes a character and moves the cursor left one character position.

Ctrl-I Moves the cursor to the next tab stop. Tab stops are automatically set at each eighth column. Has the same effect as pressing the tab key.

Ctrl-J Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a Ctrl-M keystroke.

Ctrl-K Deletes to the end of the line from the cursor.

Ctrl-M Sends the command line to CP/M 3 and returns the cursor to the beginning of a new line. Has the same effect as a RETURN or a Ctrl-J keystroke.

Ctrl-R Retypes the command line. Places a # at the current cursor location, moves the cursor to the next line, and retypes any partial command you typed so far.

Ctrl-U Discards all the characters in the command line, places a # at the current cursor position, and moves the cursor to the next command line. However, you can use a Ctrl-W to recall any characters that were to the left of the cursor when you pressed Ctrl-U.

Ctrl-W Recalls and displays previously entered command line both at the operating system level and within executing programs, if Ctrl-W is the first character entered after the prompt. Ctrl-J, Ctrl-M, Ctrl-U, and RETURN define the command line you can recall. If the command line contains characters, Ctrl-W moves the cursor to the end of the command line. If you press RETURN, CP/M 3 executes the recalled command.

Ctrl-X Discards all the characters left of the cursor and moves the cursor to the beginning of the current line. Ctrl-X saves any characters right of the cursor.

TABLE II
Built-in Commands

DIR The DIR command displays the names of files and the attributes associated with the files. DIR and DIRSYS are built-in utilities; DIR with options is a transient utility.

ERASE The ERASE command removes one or more files from a disk's directory in the current user number. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA.

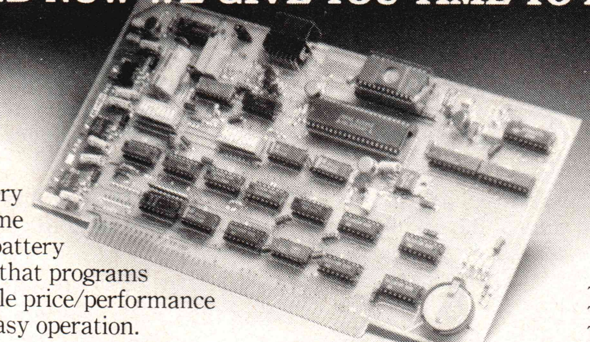
RENAME The RENAME command lets you change the name of a file that is cataloged in the directory of a disk. It also lets you change several filenames if you use wildcards in the filespecs. You can abbreviate RENAME to REN.

TYPE The TYPE command displays the contents of an ASCII character file on your screen. The PAGE option displays the console listing in paginal mode, which means that the console listing stops automatically after listing N lines of text, where N is usually the system default of 24 lines per page.

USER The USER command sets the current user number. When you start CP/M 3, 0 is the current user number. You can use a USER command to change the current user number to another in the range 0-15.

(Table III begins on page 78)

WE GAVE YOUR DRIVES THE FIRST BREAK THEY EVER HAD AND NOW WE GIVE YOU TIME TO BURN...



TimeEPROMmer, the S-100 CP/M* compatible programmer that's useful every second of every day. A real time calendar/clock with lithium battery and an EPROM programmer that programs all popular eproms. Unbeatable price/performance ratio. Features designed for easy operation.

Eprom Programmer: Port addressable. Read, Verify, Program, and Disk transfer. Handles up to 28 pins. Power generated and controlled on board. All software and documentation included. Assembled units tested with burn in.

Real Time Calendar/Clock: Complete time counting functions with CMOS LSI. Allows up to 6 months power down use. Independently port addressable.

TimEPROMmer BB & software & manual $75
TimEPROMmer Kit & software & manual $195
TimEPROMmer A & T software & manual ... $295

Our DISK CONTROL UNIT that turns 8" drives off when not being accessed. State drive.

DCU, kit & manual $29.95
DCU, A & T & manual $49.95

P & H $2. NYS add tax.
CP/M is T.M. of Digital Research

OPTRONICS TECHNOLOGY
P.O. Box 81 Pittsford, NY 14534 (716) 377-0369

Circle no. **58** on reader service card.

Overbeek Enterprises is committed to providing quality software at low prices. We intend to make a handsome profit by having thousands of satisfied customers.

Menu-Plus

A fast, convenient menu package for CP/M systems

Are your users finding it painful to cope with CP/M?

You've probably heard about the glories of menu driven systems. This powerful package developed by Capacity Inc. makes the ease of menu driven systems affordable to any CP/M user. Menu-Plus allows rapid, easy configuration of simple or hierarchical menus. You can hide the complexities of CP/M and allow single keystroke invocation of your programs. It's a user's dream. Both beginners and experienced operators will find Menu-Plus a significant enhancement. You do not have to be a programmer to install or tailor Menu-Plus on your system. In just a few minutes, you can easily create whatever new menus you desire with a text editor. Our competitors offer products in the $75-150 range. We certainly invite comparison of this product with any comparable system in terms of features or user friendliness. In terms of price there is no comparison.

$29.95

Each of these packages represents a true bargain. If for any reason you cannot install a package successfully on your system, or if you find that a package does not meet your needs, we will refund the $29.95. (Offer holds for 30 days from date of shipment.)

Micro-WYL

A powerful Z80 CP/M editor

Tired of trying to use ED under CP/M? This is the editor developed by Realworld Software, Inc. and reviewed in Infoworld (11/15/82). Here are just a few unsolicited quotes from our customers:

"Micro-WYL is undoubtedly the hottest bargain on the market."

"Thank you, thank you, thank you."

"This editor is perfect for writing in nearly any programming language. [I]... find myself looking for excuses to use Micro-WYL, and certainly have no hesitation in recommending it to anyone whose requirements match the capabilities of this inventive piece of software."

– From a review in Infoworld (11/15/82)

Now you can have the convenience of WYLBUR on your micro. CP/M is a registered trademark of Digital Research, Inc.

Disk Inspector

A full screen editor for disks that runs on Z80 CP/M systems

Have you ever been unable to read a file due to a bad sector? Have you ever erased the wrong file? Disk Inspector acts as a full-screen editor for diskettes. You can simply watch as sectors are displayed on the screen in both character and hex formats. When you wish to make the display pause, touch the spacebar. If you wish to alter a sector, it is a simple matter to move the cursor over the appropriate character, alter it, and have the sector rewritten.

Recover an erased file. Modify a directory entry.
Clean up a directory. Utilize the CP/M Auto-Load feature.
Create multiple directory entries.
Read and modify non CP/M diskettes.
Note: Disk Inspector requires an 80x24 screen on your CRT

☐ 8" SSSD
☐ Apple/Softcard
☐ KAYPRO II 5"
☐ NEC 5"
☐ Zerox 820
☐ ALTOS Series 5
☐ Televideo TS-802
☐ Osborne
☐ Superbrain
Northstar 5" DD
☐ Advantage
☐ Horizon
☐ Morrow Micro Decision

Amount: $29.95 for Menu-Plus ______
$29.95 for Micro-WYL ______
$29.95 for Disk Inspector ______
$2 for postage & handling ______
Total ______

Name________________
Address______________
City__________ State______ Zip______

Make your check payable to:
Overbeek Enterprises, P.O. Box 726, Elgin IL 60120
To order C.O.D. or with a MasterCard or Visa call
312-697-8420 between 9am and 5pm (CST)

Circle no. **59** on reader service card.

TABLE III
Utilities

COPYSYS The COPYSYS command copies the CP/M 3 system from a CP/M 3 system disk to another disk with the same format as the original system disk. For example, if the system disk is single density, the disk you copy onto must also be in single-density format.

DATE The DATE command is a transient utility that lets you display and set the date and time of day. When you start CP/M 3, the date and time are set to the creation date of your CP/M 3 system. Use DATE to change this initial value to the current date and time.

DEVICE The DEVICE command is a transient utility that displays current assignments of logical devices and the names of physical devices. This command also sets the communications protocol and speed of a peripheral device.

DUMP DUMP displays the contents of a file in hexadecimal and ASCII format.

ED The ED utility is a line-oriented context editor. This means that you create and change character files line-by-line, or by referencing individual characters within a line.

GENCOM The GENCOM command is a transient utility that creates a special COM file with attached RSX files. RSX files are used as Resident System eXtensions. GENCOM places a special header at the beginning of the output program file to indicate to the system that RSX loading is required. It can also set a flag to keep the program loader active.

GET The GET command is a transient utility that directs CP/M 3 to take console input from a file. The file can contain CP/M 3 system commands and/or input for a user program. If you use the SYSTEM option, GET immediately takes the next system command from the file.

HELP The HELP command is a transient utility that provides information for all of the CP/M 3 commands described in the manual. In the distributed CP/M 3 system, HELP presents *general* information on a command as a topic, and *detailed* information as a subtopic. HELP with no command tail displays a list of all the available topics. HELP with a topic in the command tail displays information about that topic, followed by any available subtopics. HELP with a topic and a subtopic displays information about the specific subtopic.

HEXCOM The HEXCOM command is a transient utility that generates a command file (filetype COM) from a hex input file. It names the output file with the same filename as the input file but with filetype COM. HEXCOM always looks for a file with filetype HEX.

INITDIR The INITDIR command can initialize a disk directory to allow date and time stamping of files on that disk or remove date and time stamps.

LIB The LIB command is used to create and maintain a library of relocatable object modules. Use the LIB utility to create libraries and to append, replace, select, or delete modules from an existing library. You can also use LIB to obtain information about the contents of library files.

LINK The LINK command combines relocatable object modules such as those produced from RMAC into a .COM file ready for execution. Relocatable files can contain external references and publics. Relocatable files can reference modules in library files. LINK searches the library files and includes the referenced modules in the output file.

MAC The MAC utility assembles .ASM disk files into .HEX object files. It also has the ability to process macros.

PATCH The PATCH command displays or installs patch N to the CP/M 3 system or command files.

PIP PIP is a transient utility that copies one or more files from one disk and/or user number to another. PIP can rename a file after copying it; combine two or more files into one file; and copy a character file from one disk to the printer or other auxiliary logical output device. PIP can create a file on disk of input from the console or other logical input device. PIP can also transfer data from a logical input device to a logical output device, thus the name Peripheral Interchange Program.

PUT The PUT command is a transient utility that lets you direct console output or printer output to a file. PUT allows you to direct the system to put console output or printer output to a file for the next system command or user program entered at the console. Or PUT directs all subsequent console or printer output to a file when you include the SYSTEM option.

RMAC The RMAC utility is operationally comparable to MAC except its output is in .REL form.

SAVE The SAVE command copies the contents of memory to a disk file. To use the SAVE utility, first issue the SAVE command, then run your program which reads a file into memory. When your program exits, it exits to the SAVE utility. The SAVE utility prompts you for the filespec to which the memory is to be copied, and the beginning and ending address of the memory to be saved.

SET The SET command initiates password protection and time stamping of files in the CP/M 3 system. It also sets file and device attributes, such as the Read-Only, SYS, and user-definable attributes. It lets you label a disk and password-protect the label.

SETDEF The SETDEF command lets you display or define the disk search order, the temporary drive, and the filetype search order.

SHOW The SHOW utility displays the disk drive attributes.

SID The SID utility replaces the DDT program. It offers more commands and flexibility.

SUBMIT The SUBMIT command lets you execute a group or batch of commands from a SUB file, which is a file with filetype of SUB.

TYPE The TYPE utility displays the contents of an ASCII disk file on your screen. A paging option is now available.

XREF The XREF utility provides a cross-reference summary of variable usage in an assembler program.

16-BIT SOFTWARE TOOLBOX

by Ray Duncan

MS-DOS vs. CP/M-86

Jim Howell of San Jose, California, has written me a couple of interesting letters regarding my coverage of the two major operating systems for the 8086/88 microcomputer family. I have taken the liberty of editing some of his comments and my replies into the form of a dialogue for this column, and hope he will forgive me for the slight rearrangements and changes of wording that were necessary.

JH: In the December issue of *DDJ*, you say that the reason for "excessive" coverage of the 8086/88 is a table that indicates more 8086/88s in use than other 16-bit processors. First, the numbers in that table do not necessarily indicate the level of interest in the various processors. For example, I am sure there are many, including myself, who do not have a 68000-based computer, but are nevertheless interested in that processor. In any case, if you are going to use the numbers in that table to justify the contents of your column, you should be devoting 78% of your column to the 8086/88, 15% to the 68000, and 7% to the Z8000. (What about the TI-9900 and National's 16000?).

Second, if you are going to devote 100% of your column to the 8086/88 (which is the case so far), perhaps the column should be called "8086/88 Toolbox" or even "IBM PC Toolbox." The title "16-Bit Software Toolbox" promises broader coverage than just a single processor. Actually, I suspect that the real reason for covering only the 8086/88 is that you own (or have a lot of access to) an IBM PC and have little or no access to systems that use other 16-bit processors.

RD: You certainly have penetrated right to the heart of the matter, since it is true that until very recently I didn't have any access to a 68000 system. But again, this is relevant to the content of the column and is not a coincidence. The long-range objective of the "16-Bit Software Toolbox" is to provide useful subroutines and utilities for the most commonly used 16-bit microprocessors. Although Motorola's 68000 architecture and instruction set are elegant, their decision to make a total break with the architecture of their 8-bit microprocessor family has caused a tremendous lag in the appearance of affordable operating systems and software development tools for 68000-based personal computers.

I do plan to provide increasing coverage of the 68000 in the coming months, and now have both a Cromemco Dual Z80/68000 CPU and a Godbout 68000 CPU to work with. But I'm still waiting for my copy of CP/M-68K, which was announced and demonstrated at the CP/M-83 show in January but is still not being shipped to customers.

JH: I have noticed that your recent columns have criticized various aspects of MS-DOS (apparently you are "pushing" CP/M-86, for some reason). We have used an IBM PC at work for the past several months, using only PC-DOS, and have had no problems with it.

RD: I do not have any vested interest in either CP/M-86 or MS-DOS, and am not trying to "push" one operating system in preference to another. Since MS-DOS presently dominates the 8086/88 user base, I do feel that its problems (and ways of dealing with them) should receive more space in the column. It is true that I feel the Digital Research family of operating systems and language compilers is much more "robust" than the Microsoft products; this is only an opinion, but it is based on many published comparisons and compiler benchmarks, and on my own extensive experience implementing systems tools on CP/M, CP/M-86, and MS-DOS.

JH: Benchmarks published in the November 1982 column showed PC-DOS outperforming CP/M-86 on two out of three disk-intensive tests. This was followed immediately by several paragraphs entitled "And Now the Bad News" about how changing disks in the middle of an operation under PC-DOS could clobber data on a disk, but with CP/M-86 all you get is a harmless little error message. You neglected to state that any editing you may have done to the file under CP/M-86 is lost. Perhaps that seemed too obvious to mention, but it would have balanced the presentation a little.

In January 1983, a letter from a reader appeared on the above subject and on how to recover most of the data on the damaged disk. This section had the rather negative title: "Coping with PC-DOS." The most eye-catching line in Isaac Davidian's letter was: "Unfortunately, I had no backup. . . ." If this reader had had a backup, he would have had much less of a problem. I have not used CP/M-86, but I used CP/M-80 at a previous job and found its READ-ONLY DISK error rather annoying. I always wondered why the system couldn't automatically do whatever was required to make the disk not read-only, instead of making the user start over. CP/M-86 apparently handles new disks as READ-ONLY also.

RD: Mr. Howell correctly points out that, if you switch disks at an inappropriate time under CP/M-86, you will lose the contents of the file you are currently working on. However, this is all you will lose. The integrity of the other files on both the new and previous disks is carefully protected by the operating system. The inexcusable part of PC-DOS's handling of new disks is not that you can lose your currently accessed file, but you can lose *all* of the files on the new disk; and the directory is trashed in such a manner that it is nearly impossible to reconstruct it.

This is in direct contradiction to the PC-DOS manual, page D-12, under Function 10H Close File, which states: "This function must be called after file writes to insure [sic] all directory information is updated. On entry, DS:DX point to an opened FCB. The disk directory is searched and if the file is found, its position is compared with that kept in the FCB. If the file is not found in its correct position in the directory, it is assumed the diskette was changed and [register] AL returns X'FF'. Otherwise, the directory is updated to reflect the status in the FCB and AL returns 00." If PC-DOS *really* handled file closure the way the manual states, everything would be peachy.

As for Mr. Davidian's lack of backups, we are all human and I am sure we have all failed at one time or another to make backup disks as often as would be desirable. The operating system ought to help protect us from our frailties, not exploit them!

JH: My own feelings on CP/M-86 vs. PC-DOS are based on Dave Cortesi's column of several months ago, on my four years of working with CP/M-80, and on six months of working with PC-DOS. CP/M-86 is, as I understand, not much more than a direct copy of CP/M-80. PC-DOS is based on CP/M-80, but not as heavily as CP/M-86. Microsoft and previous authors attempted, with PC-DOS, to make some improvements in moving to the 8086/88, such as:

- Time and date stamping of disk files.
- Changing the name PIP to COPY and making it a built-in command.
- Putting the parameters of COPY and RENAME in the "right" order.
- Allowing disk read and write in blocks

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems

EDITORIAL DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________________ Address ______________________

City ______________________ State __________ Zip __________ May 1983 #79

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
Articles:	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256		

Profession:
1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:
1. ______________
2. ______________
3. ______________

Purchase of magazine:
1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 756 MENLO PARK, CA
POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal
For Users of Small Computer Systems

P.O. BOX E
MENLO PARK, CA 94025

Dr. Dobb's Journal
For Users of Small Computer Systems

Reader Service Card

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. This card is valid for 90 days from issue date.

Name ______________________ Address ______________________

City ______________________ State __________ Zip __________ May 1983 #79

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
Articles: 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256

Profession:
1☐Programmer/Analyst
2☐Engineer/Scientist/Technician
3☐Business Owner/Manager
4☐Educator
5☐Consultant
6☐Professional (Law, Medicine, Accounting, etc.)
7☐Student
8☐Other ______________

Brand name of microcomputer you work with:
1. ______________
2. ______________
3. ______________

Purchase of magazine:
1☐Subscription
2☐Computer Store
3☐Newsstand
4☐Bookstore
5☐Passed on by friend/colleague
6☐Other ______________

Comments:

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

PLACE STAMP HERE
The Post Office will not deliver mail without postage

Dr. Dobb's Journal
For Users of Small Computer Systems
ADVERTISING DEPARTMENT
P.O. BOX E
MENLO PARK, CA 94025

PLACE STAMP HERE
The Post Office will not deliver mail without postage

DDJ Begins Where The Others Leave Off

12 issues for just $25.

☐ **YES!** Please send me the next 12 issues of *Dr. Dobb's Journal* for $25.00. I save $5.00 off newsstand price.

☐ Please bill me ☐ I enclose check/money order

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

Card # ____________ Exp. Date ____________

Signature ____________

Name ____________

Address ____________

City ____________ State ______ Zip ______

Offer good in USA only. Foreign rates upon request.

Dr. Dobb's Journal
For Users of Small Computer Systems
A Publication of People's Computer Company

P.O. Box E
Menlo Park, CA 94025 (415) 323-3111

S1

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps, or for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond. Simply fill it out and drop in it the mail. We take care of the rest.

Thanks for taking a few minutes to talk with us.

– Editor

Which articles or departments did you enjoy the most this month ? Why ?

____

____

Comments or suggestions:

____

____

____

____

Circle **no. 83** on reader service card.

Circle **no. 5** on reader service card.

Circle **no. 64** on reader service card.

[record sizes] of anything the user wishes, not just 128 bytes as with CP/M. There is still much room for improvement here, but it sometimes will allow reading or writing a file with one call to the system (not counting open and close), instead of having to write 128 bytes at a time in some sort of a loop.

- Showing the exact size of the file in the directory listing.
- The ability to use device names, such as CON: and LPT1: as the "filename" in an FCB, which gives some degree of device independence. Again, there is room for more improvement here, since real device independence is what one would really like.
- The batch facility of PC-DOS is much better than CP/M's SUBMIT, which is a real kludge. For example, the AUTOEXEC facility of PC-DOS allows executing a program or series of programs on power up. We use this at work to read the date and time from a battery-powered clock to set the system's date and time. Doing this under CP/M-86 requires, as far as I know, modifying CP/M's boot file (or giving the appropriate command manually each time the system is booted).

RD: Points granted, with the following reservations: Concurrent CP/M and the new CP/M Plus support time and date stamping of files, and additionally offer the convenience of password protection and "user areas." The new CP/Ms also allow file access in "burst mode" where up to sixteen 128-byte records can be transferred with a single operating system call, but this is still much weaker than the PC-DOS block read/write functions where the record size can be from 1 to 65,535 bytes. Concurrent CP/M has the capability to execute a user-supplied batch file at cold start.

Whether or not PC-DOS's COPY and RENAME parameters are in the "right" order is arguable. CP/M's format (which mimics an assignment statement) certainly seems natural to programmers, but PC-DOS's conventions are easier for new computer users to remember.

In general, I think we will all benefit from the intense rivalry between MS-DOS (and its clones) and CP/M-86. Both systems already show significant improvements over their 8-bit counterparts, and the competition for market share is bound to bring us many additional enhancements in future versions. MS-DOS 2.0, which is scheduled for release soon, incorporates some sophisticated new features including the ability to load user-designed peripheral device drivers without patching the operating system.

IBM PC Character Set Linker

Patrick Banchy, of New York City, wrote in with several suggested modifications to the CLINK utility that was published in an earlier column.

In order to get the program to work at all, when assembled with the Microsoft Assembler to form an EXE file, several pseudo-commands to define segment start/ends must be inserted. Additionally, the instruction:

MOV 14 [BX],1024

must be modified to:

MOV WORD PTR 14 [BX],1024

(the Microsoft Assembler does not check the magnitude of the immediate data to determine whether a 16-bit transfer is necessary). On his system, instructions also had to be added to intialize byte 32 of the file control block to zero before the file access, although on my PC this

happens automatically when the program is loaded.

Patrick then made some improvements which can be summarized as follows:

All segment registers of a COM program are set to the start of the Program Segment Prefix (PSP), so the first three instructions of CLINK as listed in *DDJ* are unnecessary.

DOS function #37 can be used to set the interrupt vector address.

To decrease the amount of memory made unavailable for use by other programs, he put the code that reads in the file after an ORG 480H statement, and used the default disk transfer address (80H). This results in a bigger load module, but smaller memory loss (1152 bytes for my version vs. 1424 bytes for Patrick's).

Finally, he noted that if CLINK is invoked multiple times, it will reserve a new block of memory each time. This could become quite wasteful, so he changed CLINK to check the contents of the vector location at 007CH. If the link is zero, CLINK reads the character (INT 27H); if the link is nonzero, CLINK points the disk transfer address to the previously loaded table area, reads the new table, and then makes a "normal" exit (DOS function 0).

FLIP Utility for the IBM PC

Simson L. Garfinkel sent in a program listing with the following comments:

"I own an IBM PC with both the color/graphics and monochrome adapters installed. IBM makes no effort to support users who have both of these interfaces installed, other than a brief note on page I-8 of the BASIC manual on how to switch from one display to another.

"I have written a small program (see listing on opposite page) which flips from one display to the other. The program is designed to be converted into a COM file with EXE2BIN program, and for this reason has no stack segment (this generates an error when linked which should be ignored).

"When FLIP.COM is executed, if the monochrome display was being used, the color screen is cleared and becomes the system display, and conversely."

Users who have a high-resolution color monitor will probably want to alter the program slightly so that it selects the 80x25 BW text mode when selecting the graphics interface.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 241**

Program FLIP

```
comment $

        PROGRAM FLIP

        This program flips from the color-adapter to the monochrome-adapter
        and back again.  The program has no stack segment so that it can be
        converted into a COM file.  This generates an error upon linking which
        should be ignored.

        Simson L. Garfinkel, 1983

        $

data    segment at 40h
        org 16
equip   db ?
data    ends

cseg    segment para 'code'
        org  100h                  ;I want to use this as a COM file
start   proc far
        assume cs:cseg,ss:cseg,ds:data '

        mov  ax,cs
        mov  ss,ax
        mov  sp,stacke             ;get end of stack

        push ds
        xor  ax,ax
        push ax                    ;save how to get home
        mov  ax,data               ;use ds to reference segement at 0
        mov  ds,ax
        mov  ah,equip
        and  ah,30h
        cmp  ah,30h                ;check to see if we are running mono or color
        jne  mono
        jmp  color                 ;we are in mono - switch to color
mono:   or   equip,30h             ;switch to monochrome interface
        mov  ax,2
        int  10h                   ;generate interrupt to reset board
        mov  ah,1
        mov  cx,12*256+13          ;set cursor for lines 12 + 13
        int  10h                   ;
        ret                        ;go back to ms-dos
color:  and  equip,0cfh
        or   equip,010h            ;switch to color interface
        mov  ax,4                  ;clear graphics memory
        int  10h
        mov  al,0
        int  10h                   ;select 40x25 BW text, no color burst
        mov  ah,1
        mov  cx,6*256+7            ;set cursor for lines 6 & 7
        int  10h
        ret
start   endp
        db 16 dup (?)              ;stack area
stacke  label near                 ;End of stack
cseg    ends
        end start
```

SBC, TSX and TXS

(Continued from page 68)

6502. These instructions include SBC, TSX, and TXS, and it is important that those using both the 6800 and 6502 understand the small but important differences in these instructions.

References

[1] Leventhal, Lance R. *6502 Assembly Language Programming*, Osborne/McGraw-Hill, 1979, p. 3-107.

[2] Tan, B.T.G. "Common instructions of the 6800 and 6502," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, Volume 4, Number 9, October 1979, pp. 38-39.

[3] Merrin, Stephen. "Addition and Subtraction: The 1802 Versus the Z80," *Byte*, Volume 6, Number 3, March 1981, pp. 224-228.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 237**

You've spent thousands on your system.

Will you invest two dollars to make the most of it?

OF COURSE YOU WILL.*

Computers: A Comprehensive Book Guide is a 64-page annotated mail order catalog of the best microcomputer books published to date. It critically reviews 819 books chosen to help you make the most of your microcomputer. **Computers: A Comprehensive Book Guide** is organized into 26 topics including:

- Business Applications
- Word Processing
- Assembly language & microprocessors
- Machine specific hardware & software
- Programming languages

TRY our fast, efficient, personal mail order service. The **Yes! Bookshop**, established 1970, stocks all the books reviewed. We also welcome special orders.

AND as a **Yes! Bookshop** customer, you will receive free updates, reviewing the latest microcomputer books as they are published.

WHO SAID MICROCOMPUTERS WOULD MAKE BOOKS OBSOLETE?

*Here's my $2 (refunded with first order).
Please send **Computers: A Comprehensive Book Guide** to:

Name ______________________

Address ______________________

______________________ Zip

Yes! Bookshop
1035-A2 31st Street N.W.
Washington, D.C. 20007

Mail Orders: (202) 338-2727
M-F: 9-5
VISA, MC, AMEX

by Michael Wiesenberg

Mice Are Cheaper by the Thousands

Lisa has a mouse. Xerox has been using mice for years, particularly with SmallTalk. Do *you* need a mouse? The **3G Micromouse** has two pushbuttons and rolls on a table top, moving the cursor on your screen correspondingly. You'll have to do the software interface, but 3G will sell you the hardware for $180, or $72 each in lots of over 1000. **Reader Service No. 101.**

16 Bits for Less

The **Sage II** 68000-based microcomputer with 128K RAM, running at 8MHz and 2MIPS (million instructions per second), with one 640K floppy, and p-System OS costs less than $3600. It's a small 3.9- by 12.5- by 16.7-inch box to which you'll need to add a terminal. Expansion RAM, claims Sage, is at the industry's lowest available price, $250 per 128K, and you can fit another, optional floppy into the box. CP/M-68K is another available option. **Reader Service No. 103.**

Dynalogic's Hyperion is a truly portable, self-contained 16-bit computer that is compatible with the IBM PC running MS-DOS, and costs under $3400. The "entry-level" 18-pound Hyperion has 256K RAM, one 320K 5¼-inch drive (that reads both single- and double-density IBM PC diskettes), MS-DOS, Microsoft Advanced Disk BASIC with graphics, a 7-inch non-glare amber phosphor screen of 80 columns by 25 lines and five pages of display memory, 256 display characters, bidirectional scrolling, soft-key labels on the 25th display line, serial and parallel ports, and a low-profile, detachable keyboard with 84 keys, including ten function keys and numeric keypad and optional audible key click. Options include a second floppy drive for $650; a 300-baud, built-in, direct-connect Bell 103J compatible modem with autoanswer and autodial for $495; Dynalogic's IN:SCRIBE text editor for $175; Microsoft's Multiplan spreadsheet for $257; and a soft vinyl travelling case with accessory pockets for $90. These options are standard in the $4995 **Hyperion Plus**, which also includes a time and date clock with battery backup that keeps the time even when AC power is removed. You can also get Microsoft's BASIC compiler, COBOL, Fortran, and Pascal. **Reader Service No. 105.**

What Looks Like an Apple? (But Costs Less)

Microcomputer II from **HSC Computer Services** is an Apple-clone for $499 (plus $15 p&h) that runs AppleSoft (and Franklin Ace) software and comes with 64K RAM, 6502 CPU, built-in RF modulator, high-res graphics, cassette, printer, and cartridge interfaces, AppleSoft-compatible BASIC, self-diagnosis cassette, manuals, real keyboard, and switching power supply. Options include printer, joystick, 12-inch green-phosphor monitor, speech synthesizer, Pascal, Forth, and assembly language cartridges, inventory control and accounting cartridges, RS-232C interface, floppy disk drive, and disk controller. **Reader Service No. 107.**

Phone-y Modem

Datamate 103 from **Cermetek Microelectronics** is an intelligent, 300-baud, autodial, full-duplex modem that comes with slim-line telephone receiver, for voice and data communications. It attaches to your computer's RS-232C port, stores the last data number and six other numbers in its nonvolatile memory, makes voice calls even when the modem is turned off, dials from memory, and dials until answered. The package is $295, or modem phone alone for $29.95 and Datamate 103 for $269. They've also got the same package in Bell-212A-compatible, 1200-baud model. **Reader Service No. 111.**

IBM Talks to Phone

I've mentioned **PConnection**, from **Microperipheral**, before: a plug-in IBM PC modem card – direct connect, Bell 103/113 compatible, with autodial (Touch Tone or pulse) – that answers automatically in both originate and answer modes, and fits inside the PC

IBM Clone

Vitek will be distributing **Corona Data Systems**' new IBM-compatible desktop and portable personal computers, that sell for about a third less than the "real thing." You get a green-phosphor monitor with 640-by-325 high-resolution graphics, disk drive, four card slots (3½ in the portable model), 128K RAM (expandable to 512), serial and parallel ports, and detachable keyboard. **Reader Service No. 109.**

cabinet. It used to cost $350. Now they've added automatic disconnect circuitry for failure or carrier loss, and cut the price to $279. They've also added to their line an enhanced version with real-time clock for auto-answer and autodial at preset times, and an additional serial I/O port to increase the PC's communications capabilities, for $350. **Reader Service No. 113.**

Zenith Talks to IBM

Zenith's 3270 Terminal Emulation package for Z-89 and Z-90 desktop computers includes software and bisynchronous interface card for $750, or software alone for $650. With a printer, the program emulates the IBM 3276 Model 2 with 3278 printer. Zenith's function keys are used in the same way as those of the 3270 series, with color-coded labels included to adapt keyboards to those of the 3270. You need a modem with synchronous RS-232C interface and 64K RAM. **Reader Service No. 115.**

Apple and IBM Are Now Speaking to Each Other

Direct Connect from **Trax** transfers files between Apple II and IBM PC, requiring no extra hardware (beyond a cable that they supply). This software product uses the PC's cassette port and the Apple's I/O, with transfer rates of 10K bps and automatic error checking. A "remote commander" feature controls both machines from the PC, using only one display, and permitting running of Apple programs with standard keyboard input and display on the PC. You also get a Pascal interface for your $170. **Reader Service No. 117.**

Fortran 400 Times Faster

Do you have a CompuPro 8085/88 with Hudson and Associates' 8087 Support Board? If so, you can relink your 8-bit software, including Fortran-80, COBOL-80, BASCOM-80, and MACRO-80, with F87LIB.REL, part of **Avant-Code's Fortran-87**, and run programs up to 400 times faster by taking advantage of the 8087's 80-bit floating-point routines. $200 gets you an 8-inch CP/M disk and manual, and, if you don't have the Hudson board (including 5MHz 8087-3) – installation of which does not void CompuPro's CPU warranty – you can get software and board for $695. **Reader Service No. 119.**

Beat the Computer

Arisoft's Mike Caro's Video Poker for Apple II includes two games, Jackpot Video Poker and Poker Flurry. The former simulates the progressive jackpot-type video poker slot machines. Although the game is functionally identical to the Las Vegas variety, it contains several added features. Any hand can be replayed. You can set the jackpot to what you like, carry the jackpot over from one playing session to the next, and ask the computer for a mathematical analysis of how you're doing. You can get an instant replay of the last hand. Arisoft describes the feature this way: "Although this won't alter your results or allow you to change your draw, it's good for settling arguments about whether you threw away a pair by accident. It also lets you relive moments of glory." You can change the speed with which the cards are dealt, from superfast to slow motion. The game is accompanied by sound effects, including a victory song when you catch a big hand.

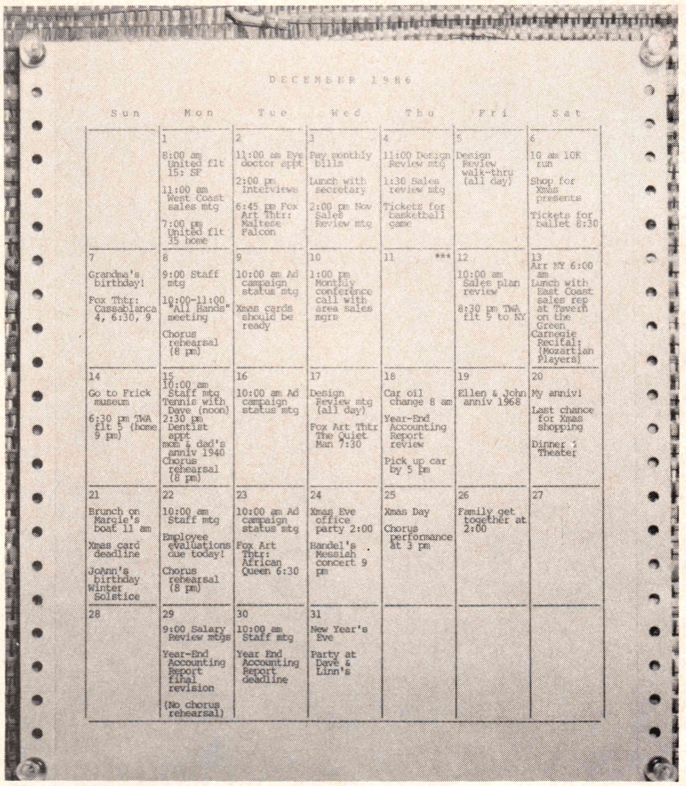
Poker Flurry is a competitive form of video poker. You can play against the computer, or against a friend. Or you can even watch the computer battle against itself! If you do this, you can also learn proper strategy for these casino games that knowledgeable players can get an edge on. (This game is the ultimate decision maker. For example, if you don't know whether to go to work today or play golf, you have the computer battle itself under the names *WORK and *GOLF, and see who wins.) There's even a doubling facility. The disk costs $39.95. **Reader Service No. 121.**

Develop Z8000 on Z80

2500 A.D. Software has a **Z8000 Cross Development** system that runs on any 64K Z80 CP/M system. You get a Z80 to Z8000 translator, 8080 to Z8000 translator, and Z8000 relocating macro-cross assembler. The latter has nested and recursive macros, include files, a linker that links up to 400 files, and relocatable code generation. All this and a 150-page manual for $179.50. **Reader Service No. 123.**

DDJ

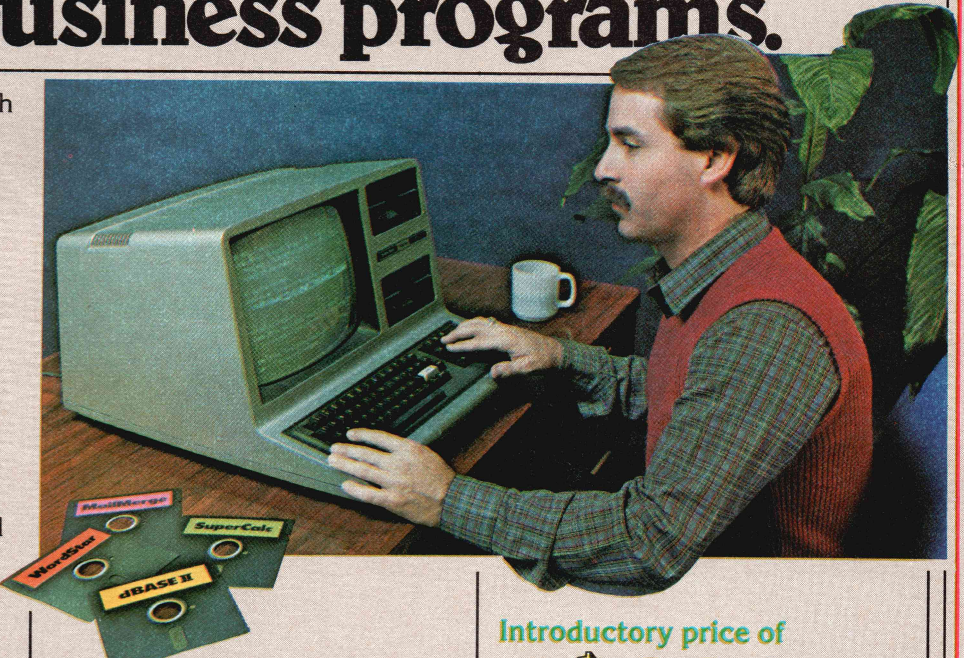
Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 243**



Print Your Own Calendars

Calendar/1 from **Clear Systems** formats calendars for terminal display or printing from text editor entered in no special order or format. It costs $60 and runs under 52K CP/M-80. **Reader Service No. 125.**

Put 64K CP/M® 2.2 in your TRS-80 Model III and tap into 2,000 business programs.



Now you can run programs such as WordStar, dBASE II, SuperCalc, MailMerge and virtually thousands of other CP/M-based programs on your TRS-80 Model III.

CP/M 2.2 is the industry standard operating system that gives you access right now to over 2,000 off-the-shelf business programs.

Our plug-in Shuffleboard III comes with 16K of RAM, giving your Model III the power of full 64K CP/M 2.2 without interference of the ROM or video memory. In fact, the Shuffleboard will appear transparent in the TRS-80 mode and will not interfere with any DOS operation.

READ and WRITE Osborne, Xerox and IBM personal computer software plus many more popular formats.

Unfortunately, there is no standardized CP/M format for 5¼" diskettes. But we have developed a way to READ/WRITE and RUN standard programs under the following single-sided formats: Osborne 1 S/D, Xerox 820 S/D, IBM PC* D/D for CP/M 86 only, Superbrain D/D, Kapro II D/D, HP 125 D/D and TeleVideo D/D.

*Will Read and Write Only.

Easy plug-in installation.

It's so simple. The Shuffleboard III plugs into two existing sockets inside your Model III. There are no permanent modifications, no cut traces and no soldering. You'll be up and running in minutes.

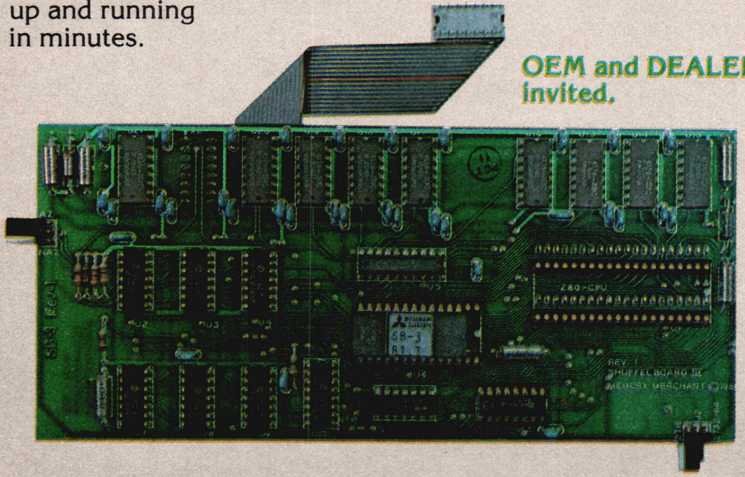
New Products.

80 × 24 VIDEO BOARD: Features dual intensity screen, programmable cursor control for block, underline & blink rate, on-board bell with audible keyclick, battery-operated real time calendar/clock, full ASCII character set plus 256 special character graphics, dual RS-232 outputs and composite video output.

FLOPPY DISK CONTROLLER: Now you can access 5¼" and 8" floppy disk drives in any combination up to 4 drives of S/D density, S/D sided. Tap into a wealth of CP/M software which comes on 8" IBM 3740 format or Pickles & Trout CP/M for the Model II.

SOFTWARE: Additional CP/M software programs are available. Call or write for details.

OEM and DEALER inquiries invited.



Introductory price of

$299.

The Shuffleboard III comes fully burned-in and tested complete with 64K CP/M 2.2 and MBASIC 80 interpreter, plus software manuals and a first class user's manual — with a 1-year limited warranty and 15-day no-risk free trial — for only $299.

See the Shuffleboard III at your dealer's now.

Once you see what the Shuffleboard can do for your Model III you'll want one at once. If your dealer does not yet stock the Shuffleboard have him give us a call. Or send check, money order, VISA or MASTERCARD number (sorry, no COD's) plus $5 shipping per board ($17 outside the USA & Canada)* directly to the address below. Cal. residents please add sales tax. Credit card purchases can be phoned in directly and we'll ship from stock.

(415) 483-1008

*Air mail shipments to Canada & all other countries.

Memory™ Merchant

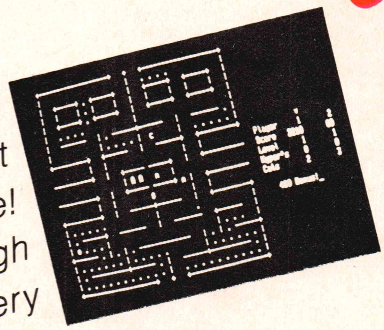
14666 Doolittle Drive San Leandro, CA 94577
(415) 483-1008

WordStar & MailMerge are trademarks of MicroPro. SuperCalc is a trademark of SORCIM. dBASE II is a trademark of Ashton-Tate. CP/M is a trademark of Digital Research. TRS-80 is a trademark of Tandy Corporation. IBM is a trademark of IBM Corporation.

Circle no. 22 on reader service card.

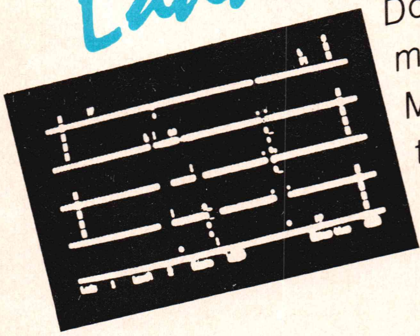
Arcade Style Games for CPM

Catchum™



Hungry little Cs chased through a maze by Ms and As, eating energy dots as they go, just like the little yellow guys you know and love! Multiple levels and screens! Keeps track of high scores! Easily personalized for virtually every terminal! And no quarters needed!

Ladder™



Dodge falling barrels as you climb to the top of this multi-level course! Just like the game with the gorilla! Multiple levels and screens! Personalized for your terminal in less than a minute! Enjoy the excitement of the popular arcade games with your CP/M computer!

Each **$24.95**

Both **$39.95**

VISA®

Please include $2 postage and handling with each order. California residents include 6% sales tax (6½% in L.A. County). Indicate 5¼″ or 8″ disk, single or double density, single or double sided, soft, 10 or 16 sectors. Call or write for complete catalogue with challenging word games like ADVENTURE and STAR TREK ADVENTURE and other entertaining and useful programs. Our offices are at 2463 McCready Avenue, Los Angeles, CA 90039. Our telephone number is (213) 661-2031.

MasterCard

CDE CALIFORNIA DIGITAL ENGINEERING

P.O. BOX 526 ★ HOLLYWOOD, CA 90028

Circle **no. 9** on reader service card.

ADVERTISERS INDEX

EXTRA

EXTRA

S-100 World News

MACROTECH International Corporation — 22133 Cohasset Street, Canoga Park, California 91303 • 213-887-5737

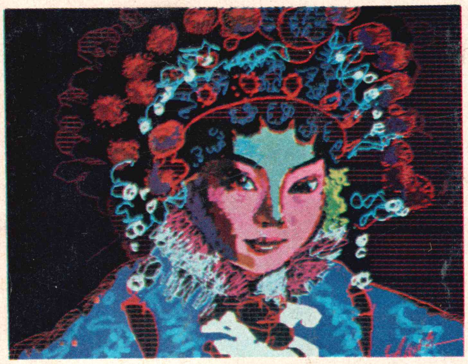
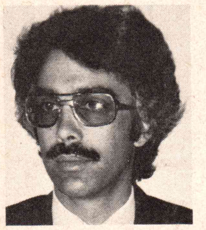


Image achieved by DGS' CAT 1600 Series color video graphic workstation. Picture courtesy of Digital Graphic Systems, Inc. See story below.

GRAPHICS: NOW MAX-IMIZED

CANOGA PARK—March 30, 1983—The decreasing costs and increasing density of memory made possible the present boom in digital graphics. Graphic systems designers are now able to take another major step with the introduction of MAX-M, a one megabyte memory board for $1983. As large size system memory and multi-megabyte Virtual Disk, MAX-M opens up major new low cost implementations.



Wayne Maw, Director of R&D for RGB Dynamics, Salt Lake City, Utah, reports, "My application is dependent on speed. With the Macrotech dynamic board, I have the needed speed." The RGB system is a Z80-based, high resolution color directory system for shopping malls, due for April release.

Empirical Research Group of Kent, Washington, creates a state-of-the-art high resolution color video graphics system by integrating their fast 68000 computer, Macrotech system memory, and the color video image processor from Digital Graphic Systems, Inc., Palo Alto, California. Radcliffe Goddard of Digital Graphics states, "High speed image processing requires large system memory to provide instantaneous display frame paging."

The demand for MAX-M by the graphics industry was nearly instantaneous following the initial Macrotech announcement.

MAX—256K to 1M S-100 Memory

CANOGA PARK—March 30, 1983—Mike Pelkey, Macrotech International president, today released details of the revolutionary MAX line of S-100 memory boards. Pelkey stated: "IEEE-696 now has a new standard for dynamic memory. The MAX product line offers 256K to 1M, at a price that ranges down to less than $0.00023 per bit." Pelkey continued, "The MI product line now includes our ultra fast (70 ns) 128K static memory, with battery backup capability, plus the 150 ns dynamic memories—in every 128K step from 256K through 1M (1024K) bytes, and add-on kits to permit field upgrade of sizes."

The extreme density of the MAX family is made possible through the use of proprietary PALs (programmable array logic). Also stated as available for add-on to any size MAX is Macrotech's popular M^3 memory mapping architecture. M^3 permits the 16-bit address space of an 8-bit processor to be dynamically mapped in 4K pages into as much as 16 megabytes of physical memory.

Parity error detection and 8/16 bit data transfer capabilities are provided as standard on the MAX series memory board.

PRICE INDEX

	SIZE	P/N	PRICE
Static Memory	128K	128-ST	$1232
Dynamic Memory 24-bit Addressing	256K	MAX-256	$1108
	384K	MAX-384	1292
	512K	MAX-512	1647
	768K	MAX-768	1815
	896K	MAX-896	1859
	1M	MAX-M	1983
With 16-bit M^3 Addressing option, add $91			
	FROM/TO	P/N	PRICE
Upgrade Kits	256K/384K	MKT-2/3	$ 192
	256K/512K	MKT-2/5	692
	256K/768K	MKT-2/7	876
	256K/896K	MKT-2/8	967
	256K/1M	MKT-2/M	1060
	384K/512K	MKT-3/5	600
	384K/768K	MKT-3/7	784
	384K/896K	MKT-3/8	876
	384K/1M	MKT-3/M	968
	512K/768K	MKT-5/7	284
	512K/896K	MKT-5/8	376
	512K/1M	MKT-5/M	468
	768K/896K	MKT-7/8	192
	768K/1M	MKT-7/M	284
	896K/1M	MKT-8/M	192
M^3 option		MKT-M3	121
Software (provided on 8" disk) Virtual Disk for MP/M II* and CP/M 2.2, CP/M 3.0* Bios modules, CP/M memory tests			$ 25
Manuals (sold separately)			
128/ST			$ 15
MAX Technical Manual			15

Software for M^3 Available

BURBANK—March 30, 1983—"M^3 bank switching for 8-bit processors is much more useful with the new creative systems programs," states Dan West of Westcom Systems Inc. MP/M II* disk intensive applications are greatly improved with the new Virtual Disk routines now available through Macrotech OEM's and dealers for their M^3 memory boards.

Westcom Systems, as the software consulting firm for Macrotech, has also provided subroutine listings to easily incorporate M^3 mapping into the new CP/M 3.0* (CP/M Plus*) Bios module. The advantages of CP/M 3.0* with disk buffering, hashed directories, and user program expansion go hand in hand with Macrotech's flexible "bank switched" memory capabilities.

All Macrotech software and manuals are available through Dan West's Compuserve account #70250,102.
Leave comments/questions as E-Mail.

These new techniques can combine the above features with custom needs of the future, such as printer buffering, multi-page display and memory-intensive graphics displays.

The software listings are included in the Macrotech memory board manuals and are optionally available on 8" diskettes.

Circle **no. 49** on reader service card.